0-315-01039-8

National Library of Canada
Canadian Theses Division
Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Division des thèses canadiennes

48897

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

ANNE BRINDLE

Date of Birth — Date de naissance
JUNE 21, 1952

Country of Birth — Lieu de naissance
U.S.A.

Permanent Address — Résidence fixe
5405 ABBOTT PLACE
EDINA, MINNESOTA 55410
USA

Title of Thesis — Titre de la thèse
GENETIC ALGORITHMS FOR FUNCTION OPTIMIZATION

University — Université
UNIVERSITY OF ALBERTA

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée
PH.D.

Year this degree conferred — Année d'obtention de ce grade
1980

Name of Supervisor — Nom du directeur de thèse
DR. JEFFREY R. SAMPSON

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

Date
August 20, 1980

Signature
Anne Brindle

NL-91 (4/77)

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ MICROFILMÉE TELLE QUE NOUS L'AVONS REÇUE

Ottawa, Canada
K1A 0N4

NL-339 (Rev. 8/80)

THE UNIVERSITY OF ALBERTA

GENETIC ALGORITHMS FOR FUNCTION OPTIMIZATION

by

© ANNE BRINDLE

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL 1980

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled "Genetic Algorithms for Function Optimization", submitted by Anne Brindle in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Jeffrey R. Sampson (Supervisor)

Francis Y. Chin

T. Anthony Marsland

Kenneth Morgan

Lenhart K. Schubert

Kenneth A. De Jong (External)

Date August 11, 1980

dedicated

to

my parents

ABSTRACT

Genetic algorithms are adaptive search procedures which employ analogies to the mechanisms of adaptation in natural evolution to provide solutions to a wide variety of search problems. This work analyzes these algorithms within the context of function optimization. First, a formal framework for search problems and methods is presented. It is seen that global function optimization is often concerned with problems arising from static complexity (e.g. multimodality) and a priori ignorance. Overall best performance is chosen as the measure which should be used to evaluate optimization methods for static, deterministic functions.

Then the various features of genetic algorithms are described in detail. Eight experiments are conducted on a set of seven multimodal test functions to determine which specific genetic algorithm is best suited to (static) function optimization. The experiments reveal that the use of larger memory than previously and the careful control of stochastic error during an internal sampling process improve algorithm performance. Representations of problem solutions which derive from the genetic phenomenon of diploidy are shown to have little effect on performance.

Finally, one experiment is designed to compare a genetic algorithm to the best methods available for global optimization of functions of unknown difficulty. Both the

genetic algorithm and a "direct search" numerical method are superior to random search. The numerical method is impaired by multimodality in Euclidean space. The genetic algorithm transforms points in Euclidean space into points in a space defined in terms of Hamming distance, and is impaired by nonlinearities in this space. There are many types of problems which do not require this transformation and have natural Hamming representations. Genetic algorithms using binary representations are probably better suited to such Hamming space problems than to functions defined over Euclidean space.

ACKNOWLEDGEMENTS

I would like to thank the members of my supervisory committee, Drs. Francis Chin, Tony Marsland, Ken Morgan, Len Schubert, and Ken De Jong, for the time and attention they have given to me and my work. A special acknowledgement is due my supervisor, Dr. Jeff Sampson, for being a guiding light, personal mentor, and protector of the English language.

Finally, my thanks to Rick, for his motivation, support, and above all, affection.

This work was supported in part by a University of Alberta Dissertation Fellowship.

TABLE OF CONTENTS

Chapter		Page
1.	Introduction	1
	1.1 A Formalism for Search	3
	1.2 Search Problems	5
	1.3 Search Algorithms	7
	1.4 Algorithm Evaluation	9
	1.5 Research Goals	12
2.	Genetic Algorithms	13
	2.1 The Reproductive Plan	14
	2.2 Implementations	16
	2.2.1 Initialization	16
	2.2.2 Parent Selection	17
	2.2.3 Offspring Generation	18
	2.2.4 Replacement Selection	22
	2.2.5 Population Size and Generation Gap	23
	2.3 Issues of Representation	24
	2.3.1 Allele Sets and Chromosome Size	24
	2.3.2 Bounds and Constraints	27
	2.3.3 Genotypic Variations	28
	2.3.4 Dynamic Genotype Organization	31
	2.3.5 The Semantics of Position	34
	2.4 The Problem of Loss of Population Variance	37
	2.5 Conclusions	38
3.	Function Optimization by Genetic Algorithms	40
	3.1 Difficult Functions	40
	3.1.1 Differentiability and Continuity	42
	3.1.2 Constraints	42
	3.1.3 Dimensionality	43
	3.1.4 Modality	43
	3.2 Design of Experiments	48
	3.3 Measures of Performance	50
	3.4 Test Function Set I	52
	3.4.1 Function 1	53
	3.4.2 Function 2	53
	3.4.3 Function 3	55
	3.4.4 Function 4	57
	3.4.5 Function 5	57
	3.4.6 Function 6	61
	3.4.7 Function 7	61
	3.5 Experiment 1: Random Search vs. the Default Algorithm	64
	3.6 Experiment 2: Population Size and Mutation Rate	71
	3.7 Conclusions	76

4. Parent Selection..... 79
4.1 Constructing the Probability Distribution..... 79
4.2 Sampling from the Distribution..... 81
4.2.1 Measures of Accuracy..... 82
4.2.2 Stochastic Sampling with Replacement..... 83
4.2.3 Deterministic Sampling..... 85
4.2.4 Remainder Stochastic Sampling with Replacement..... 88
4.2.5 Stochastic Sampling Without Replacement.... 90
4.3 Comparisons..... 91
4.3.1 Experiment 3: Sampling Methods..... 92
4.3.2 Experiment 4: Parent Selection Methods..... 93
4.4 Conclusions..... 99

5. Diploidy and Dominance..... 103
5.1 Natural Diploidy..... 103
5.1.1 The Reproductive Cycle..... 103
5.1.2 Recombination..... 104
5.1.3 Mutation..... 107
5.1.4 A Diploid Reproductive Plan..... 110
5.2 Dominance Schemes..... 111
5.3 Comparisons..... 117
5.3.1 Experiment 5: Dominance Schemes..... 117
5.3.2 Experiment 6: Dominance Change Operators... 121
5.3.3 Experiment 7: Haploidy vs. Diploidy..... 129
5.3.4 Experiment 8: Diploidy for Limited Resources..... 130
5.4 Diploidy in Retrospect..... 135
5.5 Conclusions..... 141

6. Comparison of Global Function Optimization Methods... 143
6.1 Numerical Methods..... 143
6.2 Test Function Set II: Algorithm Comparisons..... 146
6.2.1 Function 1..... 147
6.2.2 Function 2..... 149
6.2.3 Function 3..... 149
6.2.4 Function 4..... 150
6.2.5 Function 5..... 151
6.2.6 Function 6..... 151
6.2.7 Function 7..... 152
6.2.8 Function 8..... 152
6.2.9 Function 9..... 152
6.2.10 Function 10..... 153
6.3 Experiment 9: Global Optimization..... 153
6.4 Conclusions..... 167

7. Conclusions..... 168

References..... 174

Appendix A: Function Optimization Glossary..... 179

Appendix B: Bias of Remainder Stochastic Sampling Without Replacement..... 183

Appendix C: Mean Square Error of Three Sampling Methods 185

Appendix D: Global Optimization Methods..... 191

LIST OF TABLES

Table		Page
3.1	Experiment 1: Random Search vs. Default Algorithm Overall Best Performance	66
3.2	Experiment 1: Random Search vs. Default Algorithm Population Variance for Default Algorithm	66
3.3	Experiment 2: Population Size and Mutation Rate Overall Best Performance	72
3.4	Experiment 2: Population Size and Mutation Rate Average Maximum Frequency	74
3.5	Experiment 2: Population Size and Mutation Rate Fixation	75
3.6	Experiment 2: Population Size and Mutation Rate Online Performance	77
4.1	Experiment 3: Sampling Methods Mean Square Error	94
4.2	Comparison of Sampling Methods	95
4.3	Experiment 4: Parent Selection Methods Overall Best Performance	97
4.4	Experiment 4: Parent Selection Methods Average Maximum Frequency	98
5.1	Experiment 5: Dominance Schemes Overall Best Performance	118
5.2	Experiment 5: Dominance Schemes Average Maximum Frequency	119
5.3	Experiment 6: Dominance Change Operators Overall Best Performance	123
5.4	Experiment 6: Dominance Change Operators Average Maximum Frequency	124
5.5	Experiment 7: Haploidy vs. Diploidy Overall Best Performance	131
5.6	Experiment 7: Haploidy vs. Diploidy Average Maximum Frequency	134
5.7	Experiment 8: Diploidy for Limited Resources Overall Best Performance	136
6.1	Test Function Set II	148
6.2	Experiment 9: Global Optimization Overall Best Performance	155
6.3	Experiment 9: Global Optimization Overall Best Performance of DSC	156
6.4	Experiment 9: Global Optimization Overall Best Performance of GA	158
6.5	Experiment 9: Global Optimization Overall Best Performance of DSC versus GA After 2000 Function Evaluations	164

6.6 Experiment 9: Global Optimization
Overall Best Performance of DSC versus GA
After 5000 Function Evaluations 165
6.7 Experiment 9. Global Optimization
Overall Best Performance of DSC versus GA
After 10000 Function Evaluations 166

LIST OF FIGURES

Figure		Page
2.1	Examples of Crossover	20
2.2	Linkage Alteration Operators	33
3.1	Function f1e	54
3.2	Function f2e	56
3.3	Function f3	58
3.4	Function f4	59
3.5	Function f5	60
3.6	Function f6	62
3.7	Function f7	63
3.8	Experiment 1: Random Search vs. Default Algorithm Overall Best Performance	67
3.9	Experiment 1: Random Search vs. Default Algorithm Average Maximum Frequency for Default Algorithm	70
3.10	Experiment 2: Population Size and Mutation Rate Overall Best Performance of One Algorithm	78
4.1	Stochastic Sampling Method	84
4.2	Deterministic Sampling Method	86
4.3	Remainder Stochastic Sampling Method	89
4.4	Experiment 4: Parent Selection Methods Overall Best Performance of Two Methods	101
5.1	Mitotic Division, Meiotic Division, and Simulation of Meiotic Division	105
5.2	Offspring Creation for Diploids with Individual Dominance Maps	116
5.3	Experiment 6: Dominance Change Operators Average Maximum Frequency for One Scheme	125
5.4	Experiment 6: Dominance Change Operators Overall Best Performance for One Scheme	127
5.5	Experiment 7: Haploidy vs. Diploidy Overall Best Performance	132
5.6	Experiment 8: Diploidy for Limited Resources Overall Best Performance	137
6.1	Experiment 9: Global Optimization Overall Best Performance of PRS, DSC, and GA	159

CHAPTER 1

INTRODUCTION

Adaptation has long been of interest to those involved in problem solving. In fields from numerical analysis to automatic learning to process control, it is recognized that if the problem is very difficult, an adaptive search method may offer the best hope of achieving a good solution. An adaptive search procedure is defined as any method which continually modifies the direction of future search on the basis of information received during the search process. A gradient technique in function optimization, then, is an adaptive process, as is a computer program which "learns" to improve its chess playing.

Many biological systems are also adaptive in nature, including systems of evolution, neural systems, the immune system, and cell processes at the molecular level. An understanding of the techniques of natural adaptation can aid in the development of artificial systems. For example, knowledge of human learning behavior has influenced many computer learning techniques. The analysis of natural adaptation has also led to the development of genetic algorithms.

A genetic algorithm is a generalized adaptive search procedure which employs many of the structures and techniques of genetic systems. Research in genetic

algorithms was initiated by John Holland at the University of Michigan in the late 1960's. Holland (1975) first formalized the notion of adaptation by describing the features common to all adaptive systems. He then showed how the particular devices of genetic adaptation could be modelled within the formal framework to form a procedure applicable to many problem paradigms. The original specification, called a "Reproductive Plan", was actually an outline which could be implemented in several ways. Thus the Reproductive Plan actually defined a class of search procedures.

As with any new group of problem-solving techniques, two questions have to be answered about the class of genetic algorithms. First, what is the domain of problems over which genetic algorithms are successful and superior to other known methods? Second, what specific members of this class are most desirable? In general, these questions cannot be answered independently, as each algorithm may have its own area of high performance. Also, the scope of these issues is large and a single thesis, or even a lifetime of research, will not resolve them. In order to provide even partial answers, it is necessary to have a clear framework relating search problems, adaptation, and genetic algorithms.

1.1 A FORMALISM FOR SEARCH

The formal framework for adaptation developed by Holland can be extended into a formalization for search problems in general, regardless of the nature of the algorithms which might be employed. A search problem p contains the following components:

S = {s} the set of all possible solutions to the problem. This set may be infinite, even non-denumerable.

F:SxT-->R a function which maps an element s in S at time t in T into the set of real numbers. This is the evaluation function for potential solutions to the problem. It may be a threshold function, where a solution either is accepted or rejected, or a probabilistic function, where a solution may have a certain likelihood of being good. These and other possibilities will be investigated later.

As a side note, it may be argued that F need not map into the reals. For instance, if the problem is survival in a given natural environment, S could be the set of possible individual organisms, and F would be organism fitness, which has no intuitive representation in real numbers. However, any researcher attempting to compare the fitnesses of various organisms first develops a mapping from organisms to a partially ordered set, e.g. organisms to their life spans. At this point, the survival problem has been rewritten into a well-formulated search problem. This thesis is concerned

only with well-formulated problems and not with the issues involved in the development of evaluation functions.

An analysis of search methods includes these elements:

P = {p} the set of search problems of interest.

A = {a} the set of algorithms which are under consideration for use in solving the problems.

X:AxP-->R a function mapping the behavior of an algorithm on a problem into the real numbers. An algorithm "a" applied to a problem p can be characterized by

1) the set $\{s_{t_0}, s_{t_1}, \ldots, s_{t_f}\}$ of solution points searched, and

2) the costs of operation of the algorithm (the resources consumed).

The domain AxP is represented by these values.

U:A-->R the global measure of the utility of an algorithm on the problem set P. For example, U may be the weighted average of an algorithm's performance X(a,p) over all problems p in P.

Each of these four items is examined in detail below. Examples are drawn primarily from the fields of function optimization and artificial intelligence, although research into search techniques is by no means restricted to these areas. Cooper and Steinberg (1970) and Sampson (1976) are suggested references, respectively, for those two subjects.

1.2 SEARCH PROBLEMS

The features which render a search problem non-trivial can be grouped under four headings.

1. A priori ignorance is the absence of useful information about the problem. The set S may have no obvious bounds. In this case most search algorithms must arbitrarily impose limits and work within them, whether or not the desired solution is in the chosen region. For example, Samuel's (1959) program for learning the game of checkers uses 40 parameters in its evaluation of board positions. The learning process is a search for the optimal weights for the parameters in a linear equation. There is no guarantee, however, that the best program uses only those 40 parameters. Samuel admitted that this was an arbitrary and unsatisfactory restriction on the search, but could see no way to extend the limits to include all possible checkers strategies.

A priori ignorance also includes lack of information about the evaluation function F. In function optimization today, most methods are effective on unimodal surfaces only and many assume continuity and even differentiability. But a process control problem, for instance, may not yield a closed form function which can be examined for these features. The only evaluation procedure may be to implement the proposed solution in the process environment and observe results. In this case, F may or may not be unimodal. Any results from a unimodal search method would be unreliable,

since the algorithm might very well be operating on a problem it is incapable of optimizing.

In general, a priori ignorance of the nature of the evaluation function can only be overcome by an algorithm which is effective over the entire range of functions which can be characterized by any known information.

2. This ignorance concerning the problem can be differentiated from static complexity, in which the problem is known to have specific features which make it hard to search. The size of the solution set S can determine problem complexity, as can many characteristics of the evaluation function. Items which fall under the heading of static complexity have been a prime target for research in the field of function optimization, for example high dimensionality, multimodality, and discontinuity in the evaluation function. Static complexity calls for algorithms which are tailored to particular problem features.

3. When a problem's evaluation function is not simply a function of the solutions S, but also of time, the problem is said to involve dynamic ignorance. In many situations, the value of a solution changes from one time to another. The optimal amount of steam production to keep a building at room temperature on a September afternoon may not be the optimal amount four months later, or even twelve hours later. Functions which vary with respect to time require algorithms capable of tracking good solutions or handling large quantum changes in the evaluation function.

4. Dynamic uncertainty occurs when the function F is a probability function over R. The value obtained for a given solution s at a time t may be the result of a nondeterministic process, or may be perturbed by random noise. An excellent example of nondeterminism, which has been carefully analyzed by Holland (1975), is the problem of the two-armed bandit. A gambler is informed by the casino manager (who is honest) that one of two slot machines, Y and Z, pays off more frequently than the other. How should the gambler allocate his fixed stake to the two machines to optimize his total payoff? In the search formalism, S is the set of two plays, one on Y and one on Z. F(s), the payoff for each play s, is a probability function based on the two machine payoff frequencies. A good algorithm for this problem must respond to both the a priori ignorance regarding which machine is better and the nondeterministic manner in which the elements of S are given values.

1.3 SEARCH ALGORITHMS

Search problems can be categorized according to features of difficulty. A taxonomy can also be constructed for algorithms. Classification in this case is dependent upon the techniques employed in searching.

Indirect or analytical methods are usually not considered search algorithms, since they do not evaluate points in S during their operation. Instead, a priori knowledge of F and S is used in a series of computations to arrive at the desired solution. The classical methods of

function optimization, e.g. finding the minimum point of a parabola by setting the derivative to zero, are indirect methods.

Random search is the evaluation of a sequence of unrelated solutions. If the set S is sufficiently small, random search often can be refined into enumeration, where all elements of S are evaluated. Random search is the cheapest form of search and so serves as a benchmark in algorithm comparisons. It is also the best method currently known for some very difficult problems.

Adaptive search has already been defined as the use of feedback information to direct the search process. Algorithms which are adaptive in nature can be grouped according to the sources of the information employed.

I is the vector of information available from the solution previously evaluated. In a first-order method, I consists only of the measured value of the solution, F(s). In higher-order methods, I may incorporate information such as derivative values or individual components of F(s), in addition to F(s).

M is the memory of previous steps in the search process. Many algorithms use a fixed amount of space to save the history of the search in some condensed form. This memory is constantly updated with information from I and used to select new solutions for testing. Dichotomous Search in function optimization requires memory only of the previous two solutions evaluated, while "conjugate

direction" methods maintain information on the directions of the previous n searches.

G consists of "gift" information, or hints. This is not feedback material, but is input from outside the search process which is employed along with information from I and M. It is important to equalize gift information when making comparisons between methods.

1.4 ALGORITHM EVALUATION

To reiterate, the observed performance and resources required by an algorithm on a problem are evaluated by means of a function X. X combines certain cost measures with performance measures derived from the set of solutions tested during search. Measures of resources used include:

1) space costs, in terms of the room needed to store M and I in an adaptive algorithm, plus, possibly, the space required for storage of the algorithm itself, and

2) time cost, which is measured in any of: computer CPU time (if applicable), number of primitive operations performed, number of solution points searched in S, or number of invocations of the function F. Computer time is machine dependent. CPU time and counts of primitive operations are sensitive to variations in the coding of the algorithm. The number of solutions searched may hide large time expenditures, such as multiple function evaluations used to approximate derivatives at each solution point. Counting calls of F ignores such things as the time spent in updating the algorithm's memory, which may be considerable.

Nonetheless, function calls are usually assumed to be orders of magnitude more expensive than other steps in algorithm execution, so function invocations are a popular measure of time cost.

Three common measures of performance are overall best, offline, and online F values. The overall best value is just that: the value of the single best solution out of all solutions evaluated. This measure is of interest in function optimization and other problem areas where one end product is the goal of the search.

Online performance is measured by the (possibly weighted) average evaluation of all solutions examined. In a field such as process control, the evaluation of a solution consists of implementing the solution in the environment. It is desirable to improve the quality of the control while at the same time not subjecting the environment to poor, possibly damaging, control solutions. Online performance in such a case refers to the average quality of the solutions tested.

Some problems require a steady stream of solutions, as in the case of online process control, but the evaluation procedure is not equivalent to the implementation of the solution. If the process environment is simulated on a computer, solutions can be tested there, while the best solution to date at any time is employed in the real environment. This offline performance, as it is called by De Jong (1975), can be measured by the average over all time

steps of the best value up to that time.

There are many ways to incorporate cost and performance measures into the function X. Most functions reflect the conditions, if any, which terminate the search process. If the search is ongoing, X will be a function of the cost per time step and the online or offline performance. If the search terminates, it may be in consideration of one of the following three goals:

Fixed Resources. The algorithm consumes fixed amounts of resources and is evaluated solely on its performance. X is a performance measure value obtained after the time interval allotted for search has elapsed.

Fixed Performance. The algorithm continues until a certain level of performance is achieved. X becomes the cost in time and/or space which is incurred to reach the given performance threshold.

Variable Resources and Performance. In some problems resource limitations are unimportant and the range of performance values is unknown. For these, the best mode of operation would allow the search process to continue as long as advances were made in performance, that is, as long as the algorithm had not exhausted its potential for fruitful search. Cavicchio (1970) proposed the technique of stopping the search when the rate of change of performance fell below some threshold. X would then have to reflect both cost and performance.

1.5 RESEARCH GOALS

The purpose of research on a specific algorithm is to determine its usefulness on a set of problems. It is improbable that there is one method which is more efficient and more effective than all other methods on all search problems. Therefore it is up to the researcher to narrow the definition of "useful" to encompass only a limited problem set P and a reasonable function U for algorithm utility.

This work will focus on genetic algorithms as applied to two problem sets. The first set will be hard problems in (static) function optimization, for which random search has been the best algorithm found so far. The second will be a set of function optimization problems which span a large range of difficulty for standard search techniques such as descent methods. The goals in using these two groups are 1) to find good genetic algorithms for function optimization by testing various algorithm features on hard functions, and 2) to measure the robustness of genetic algorithms over a wide range of function optimization problems. Results from these investigations should lend further insights into the domain of applicability of genetic algorithms, as well as a better understanding of the algorithms themselves.

CHAPTER 2

GENETIC ALGORITHMS

Genetic algorithms are adaptive search procedures which simulate some of the processes of natural evolution. Groups of organisms often exhibit the ability to adapt to an environment (or environments) over several generations. Such adaptation at the population level can often be shown to be the result of, among other things, events in the differential transmission of genetically variable information at the molecular level. Molecular genetics involves the determination and analysis of the molecular structures of heredity, while population genetics is the mathematical study of the consequences of inheritance with respect to groups of organisms.

Genetic algorithms simulate a population of organisms over several generations. The representation and modification of the organisms is based on information from molecular genetics, while the expected behavior of the algorithms is derived using the methods of population genetics. However, genetic algorithms are intended to be problem-solving methods in fields such as function optimization. As such, they do not necessarily contribute to the various fields of genetics.

In this chapter, genetic algorithms are set within the formal framework of Chapter 1. Since the terminology of

genetics most accurately describes the features of the algorithms, this terminology prevails in the discussion.

2.1 THE REPRODUCTIVE PLAN

A genetic algorithm is adaptive. At each step in the search process, it has the following information:

I This has been simply F(s) in all genetic algorithms thus far. They are first-order methods.

M M is a <u>population</u> (group) consisting of organisms evaluated at earlier time steps. This population does not include all organisms from previous steps, only a chosen subset.

The Reproductive Plan, or outline of the cycle of operation, has five basic steps:

1. n organisms are generated to form the population.
2. n' organisms are selected according to fitness and copied to form a set of <u>parents</u>.
3. n" <u>offspring</u> are created from the set of parents by the application of <u>operators</u>.
4. g members of the population are selected for replacement. These organisms expire and are replaced by the offspring from step 3.
5. a new <u>generation</u> (population) has been formed. The cycle begins again at step 2.

A genetic algorithm can be applied to any problem represented in the following manner:

S is a set of organisms (structures). Each organism has

1) a genotype (genetic structure) consisting of chromosomes (strings) of alleles (values). Each locus (position) on a chromosome has a set of possible alleles associated with it.

2) a function D which maps the genotype of the organism onto a set of features evaluated by F. This set is called the phenotype. Thus D:genotype-->phenotype.

F is the environment within which organisms are evaluated. An evaluation results in a fitness value F(s) for an organism s.

For example, assume the problem is to find the maximum of a function of two variables, H:XxY-->R, where each variable ranges from 0 to 999, with the additional restriction that the solution's values for x and y must be integers. A solution s, say s=(905,82), can be represented as one chromosome of six loci "$a_1a_2a_3a_4a_5a_6$", or in this case "905082". Each of the loci has the same set of ten possible alleles, the digits 0 through 9. The function D maps this chromosome onto the set of integer pairs (x,y) (the phenotype) using the equations

$$x = a_3 + 10a_2 + 100a_1$$

$$y = a_6 + 10a_5 + 100a_4.$$

The fitness of a solution is simply H(x,y).

2.2 IMPLEMENTATIONS

Each of the steps in the Reproductive Plan can be implemented in several ways. Most features of the genetic algorithms studied up to now are modelled on some natural system. Arbitrary design decisions must occasionally be made, however, in the absence of information on natural systems, or when simplification of the model is necessary.

The theses of Bagley (1967), Cavicchio (1970), Hollstien (1971), Frantz (1972) and De Jong (1975) include research on specific algorithm features. Bagley compared certain genetic algorithms with non-adaptive correlation methods. Hollstien applied genetic algorithms to problems of control policy formation. Frantz observed the effects on genetic algorithms of non-linearities in the function F. Cavicchio and De Jong examined various parameter settings for genetic algorithms applied to problems in pattern classification and function optimization, respectively.

2.2.1 Initialization

The initial population has in all cases so far been generated according to a uniform random distribution over the set S of possible solutions. This could be altered if there was *a priori* information indicating that the search should be biased toward certain subsets of S.

2.2.2 Parent Selection

The success of the genetic algorithm as an adaptive search technique depends on the balance between history retention and exploration. The population M must serve both as the memory of good solutions tested and as the repository for new solution points. History retention is accomplished mainly by parent selection according to fitness. An organism is selected to be a parent with a frequency proportional to its fitness relative to the rest of the population. Offspring will resemble their parents, so good organisms are maintained through succeeding generations.

Relative fitness can be defined in terms of rank in the population (Cavicchio, 1970; Hollstien, 1971). However, this seems to lead to overemphasis of a few top organisms of possibly marginal superiority, resulting in loss of population variance. Loss of population variance has been a recurring difficulty in the implementation of genetic algorithms. It is discussed in Section 2.4.

Relative fitness is more commonly measured as proportion of total population fitness, $\frac{f(s_i)}{\sum_{s \in M} f(s)}$ (Cavicchio, 1970; Frantz, 1972; De Jong, 1975). Thus if an organism accounts for 0.10 of the total population fitness, it should be selected $0.10 \cdot n'$ times to be a parent. Organisms of less than average utility will be represented fewer times in the set of parents than those of above average fitness.

Selection according to proportion of population fitness can be accomplished by adjusting the fitnesses to compensate for possible negative values and then using the fitnesses to impose a probability distribution on the population. Parent selection is then a matter of sampling from this distribution. Sampling must be accurate; if the poorer organisms are overselected, history retention suffers, but if the better organisms are overselected, loss of population variance results. The issues involved in adjusting fitnesses and sampling are examined in detail in Chapter 4.

2.2.3 Offspring Generation

If offspring were merely copies of their parents, no new organisms would be evaluated. Search would be limited to those organisms represented in the initial population. Exploration is introduced into the algorithm through genetic operators. These operators alter the offspring so that they still resemble, but are not identical to, their parents. The mutation and crossover operators, when used together, provide for adaptive exploration.

Mutation. Point mutation consists of replacing an allele in a chromosome by a random allele from the set of possible alleles for that locus. In the example of the two dimensional function, a point mutation might change the chromosome "905082" into "905012". Loci to be mutated are chosen randomly, with a probability P_m that any one locus is mutated. This is a simplification of the situation in natural systems, where each locus may have its own frequency

of mutation, and mutation rates may be dependent on environmental factors such as radiation.

Frequent application of the mutation operator can result in randomly generated offspring which bear little resemblance to previous generations. Mutation in general hinders the history retention of adaptation and lends a degree of randomness to the search. It can be used as an instrument for exploration, but it serves a more important function. Mutation prevents the permanent disappearance of an allele from the population.

Assume that the genotype consists of a single chromosome of unique loci, each with two possible alleles, and that the entire population of size n is replaced each generation. The minimum mutation rate needed to insure the presence of an undesirable allele somewhere in the population for each locus at each generation is roughly $P_m = \frac{1}{n}$ (Holland, 1975). De Jong (1975) found that for values of P_m between $\frac{1}{4n}$ and $\frac{1}{n}$, offline performance was much the same, but that the higher mutation rates degraded online performance. In the end he settled on a mutation rate of $\frac{1}{20n}$ as optimal for online performance and acceptable for offline performance.

Crossover. Crossover introduces adaptive exploration into genetic algorithms. Crossover occurs when two chromosomes exchange segments (see Figure 2.1).

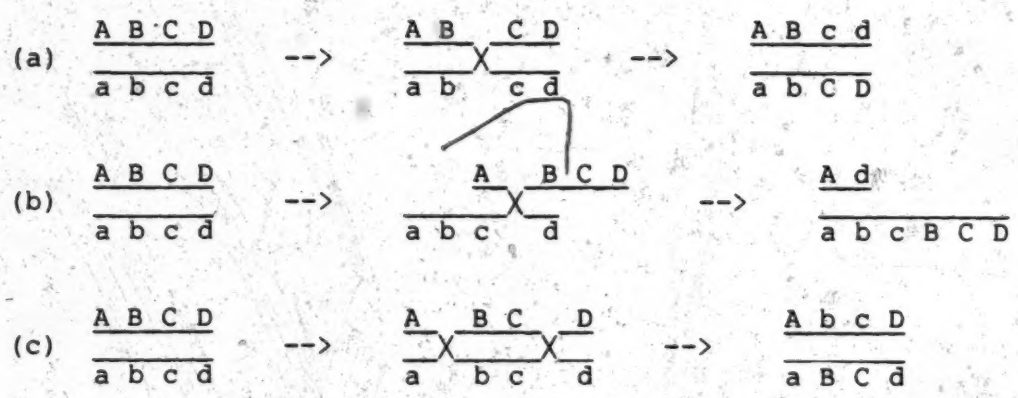


Figure 2.1. Examples of Crossover
(a) Homologous Crossover
(b) Unequal Crossover
(c) Multiple Crossover

In natural organisms, crossover usually occurs between matching sites on the two chromosomes. Let locus 1 have allele set {a,A}, locus 2 have allele set {b,B}, etc. Then Figure 2.1.a illustrates a crossover at a point of homology: both chromosomes break between loci two and three. Two chromosomes are homologous if all (or almost all) of their loci correspond, even though they do not have the same alleles at each of the loci. Crossover occasionally occurs at non-homologous points. Figure 2.1.b shows unequal crossover, where two homologous chromosomes cross at points that do not correspond. This is a possible natural mechanism for deletions and duplications of loci.

Natural chromosomes are not restricted to one cross per chromosome (see Figure 2.1.c for an example of multiple

crossover). Each point on a chromosome may have its own frequency for breaking and recombining. This frequency may be affected by environmental factors. Except for a brief examination of double crossover by Cavicchio (1970), experiments and analysis of genetic algorithms have been limited to the case of at most one crossover per chromosome pair, where pairs cross with a fixed frequency P_c. If crossover occurs, all points on the chromosome have equal probability of being the crossover point. De Jong determined experimentally that $P_c = 0.6$ optimized algorithm online and offline performance on a variety of functions.

In addition to selection according to fitness, natural adaptation relies on recombination between individuals for the development of improved populations. Crossover is an important tool for recombination. When each organism is represented by a single chromosome, crossover can occur between the chromosomes of two parents to form offspring which combine the features of both. This is analogous to the exchange of genetic information as it occurs in single chromosome bacteria. The offspring are not identical to either parent, so new organisms are evaluated, but the offspring still resemble the parents, so the exploration of new organisms is directed by knowledge of previous evaluations.

Mating. The inclusion of recombination between parents in step 3 raises questions about mating. Parental groups are limited to either one or two parents. If two parents

are paired to create offspring, there must be a mechanism for selecting the two from the set of parents. Hollstien (1971) described many natural mating schemes. He concluded that inbreeding (mating within "families", or groups of organisms with common ancestry) combined with crossbreeding (mating between the best organisms in different families) provided good performance when incorporated into a genetic algorithm. However, it is not clear that the increased expense of controlled mating is justified in light of the simplicity and good performance of random mating.

Natural systems often employ restrictions on gender in mating. Population geneticists are not sure what the advantages of having genders are; in fact, the causes and benefits of recombination and sex in natural genetic systems are still not understood (Maynard Smith, 1978). Therefore, as a simplification, this work employs random pairing (without genders) within the set of n' parents to form $\frac{n'}{2}$ parental groups. Furthermore, each parental group produces exactly one offspring, which is chosen randomly from possible offspring resulting from crossover. Thus $n'' = \frac{n'}{2}$.

2.2.4 Replacement Selection

In addition to parent selection, replacement selection influences the composition of the next generation. The predominant natural method involves the deletion of inferior organisms, or "survival of the fittest". However, in genetic algorithms the combination of small populations (50 or so), parent selection according to fitness, and

replacement selection in inverse proportion to fitness leads to loss of population variance, and thus poor performance. Cavicchio (1970) tried replacing parents with their offspring, modelling the way salmon, for instance, die immediately after mating. In general, random replacement has been used (De Jong, 1975) for simplicity. Note that if g (generation replacement size) equals n (population size), then replacement selection is no longer an issue, as the entire population is replaced.

2.2.5 Population Size and Generation Gap

Holland (1975) analyzed the expected behavior of genetic algorithms with infinite populations. But in an implementation, the size n of the population is restricted by the space resources available for the simulation. Although it is possible to have populations where the number of organisms varies up to a preset maximum, it is unclear what advantage such a scheme would provide. Thus it has been standard practice to use populations of fixed size, and set n"=g. Cavicchio (1970) and Hollstien (1971) used populations of sizes 12 to 20, but De Jong (1975) concluded that populations of under 50 suffered from loss of population variance.

De Jong also investigated different possibilities for generation gaps, that is, for g relative to n. He decided that there was no advantage to overlapping generations, where g<n. His final implementation set n=g=50.

2.3 ISSUES OF REPRESENTATION

The performance of a genetic algorithm on a problem depends heavily on the representation the problem is given. This discussion concerns the representation of function optimization problems, although many of the concepts apply elsewhere.

2.3.1 Allele Sets and Chromosome Size

A solution to a function optimization problem is an ordered set of numbers $(x_1, x_2, \ldots, x_d)$, where d is the number of dimensions in the function. In the example of Section 2.1, such a set was encoded as a chromosome composed of decimal digits. The chromosome could also have contained binary, octal, or hexadecimal digits. In the binary case, each locus would have an allele set of {0,1}. The chromosome would have to be longer than in the decimal case to represent solutions in the same solution set S.

It is not immediately obvious which is preferable, long chromosomes and small allele sets, or short chromosomes and large allele sets. Two extreme points of view are represented by models used in population genetics for the prediction of allele frequencies. The first, the infinite site model, assumes that for one organism feature there are an unlimited number of loci. Each locus may be mutated to an alternate allele, but the mutation rate is small, so the same locus will never mutate twice. In the infinite allele model, a function is encoded using one locus with an infinite number of possible alleles. It is assumed that

mutation will always introduce a non-existent allele into the population.

One point in the analysis of these two models is of interest here. Under the infinite allele model, assuming a diploid population, no recombination, and random parent selection, the expected number of alleles present at one locus in a population at equilibrium is $4nP_m + 1$, where n is the population size and P_m is the probability of mutation at each time step (Kimura, 1968).

In addition to his mathematical analysis, Kimura performed simulation experiments on a population of 100 one-locus individuals which initially contained 200 distinct alleles. In accordance with the theory, a mutation rate of 0.01 produced equilibrium populations (after about 20 generations) which contained only 5-10 alleles. Thus, even without selection according to fitness, a population with the capacity for several hundred different alleles cannot be expected to maintain more than a few alleles at any one locus. With selection, the number of alleles would shrink still further.

The infinite allele model is analogous to the situation where one x-value of a function solution is encoded using one locus on a chromosome. The only mechanism for introducing a new value into the population is mutation; crossover does not alter the alleles at individual loci. Since the number of alleles is large, only a large mutation rate will permit productive search. But as has been shown,

the result of a high mutation rate is random search.

The alternative is to use many loci to represent one x-value. The advantage is now obvious: if x_i is represented by a gene comprising several loci, crossover becomes a tool for the creation of new x-values. The more loci there are in the gene, the more crossover is involved in this task, since the probability of crossover within the gene increases. Also, if there are fewer alleles in the allele set for one locus, a smaller mutation rate will be sufficient to maintain all of those alleles in the population.

The benefit in having multiple locus genes instead of a single locus for each parameter can be summarized as an increase in adaptive power. Instead of creating new x-values only via mutation, new values are formed adaptively by recombining the smaller "building blocks" of individual loci.

For a fixed number of genic values, the maximum number of loci in a gene occurs when each locus has only two alleles. The advantage to having many loci is now apparent. Therefore, the optimal representation for function values in a genetic algorithm is binary. This is consistent with the situation in natural systems, where many individual sites with four possible values each are concatenated to form one gene.

Assume, then, that for each dimension i of a function, the minimum value $minx_i$ and the resolution between points

Δx_i are known. If x_i is represented by loci j through k on the chromosome, then the phenotype map D is

$$x_i = D(a_j,\ldots,a_k) = \min x_i + \Delta x_i \sum_{v=j}^{k} a_v 2^{k-v}.$$

2.3.2 Bounds and Constraints

No matter what allele sets are chosen, it is possible that the representation will span solutions which are not in S. If x_1 is limited to the integers between 0 and 5 inclusive and a binary representation is used, three loci will be needed. With three binary loci, eight integers can be represented, so the representation allows two solutions which are outside S.

In theory, a new representation could be chosen which takes into consideration the bounds on the search set S. In actuality it is usually not obvious how to construct this new representation, or if it is constructed, to show that it will be an effective representation for the genetic algorithm. In the same way, it is theoretically possible to build the constraints of a function optimization problem into the representation so that all represented solutions are legal. No methods exist for such transformations; the addition of constraints to a problem usually renders it unsolvable by unconstrained search methods.

The subjects of bounds and constraints will not be considered here. Experiments will be limited to unconstrained functions which have solution sets with magnitudes of even powers of two.

2.3.3 Genotypic Variations

A binary representation has one locus for each power of two. In standard notation, a binary number is written left to right in order of decreasing powers of two. However, there is no reason to assume that this ordering is superior to any other with respect to genetic algorithms, especially when several numbers are concatenated on a chromosome. Grouping the high-order loci from all dimensions together, for example, might change the effect of crossover in the search process.

Besides variations in locus order, there are many alternative formats which can be used to represent binary numbers in a genotype. In nature, genotypes range from single chromosomes with fixed orderings of the loci to multiple chromosomes, polyploid genotypes, and chromosomes of variable length. With the exception of the two studies of Bagley (1967) and Hollstien (1971), work on genetic algorithms has been limited to genotypes of a single chromosome. More complex organizations should be considered, however.

Multiple Chromosomes. Often in higher organisms the loci of the genotype are distributed on several chromosomes. These chromosomes may recombine with homologous chromosomes from another parent, but during the creation of offspring they assort independently. The resulting situation can be described as the loss of linkage between the loci on different chromosomes.

Linkage refers to the distance between two loci on a chromosome. A high linkage value indicates that the loci are close together; a value of zero means they are not on the same chromosome. In a single chromosome parent, the alleles of two adjacent loci will stay together in the offspring unless crossover occurs at the point between them. In a multiple chromosome parent, the loci on separate chromosomes are not linked and will end up in the same offspring only by chance.

Thus multiple chromosomes may be used for grouping loci whose functions are known to be independent. At the moment it is unclear whether any further benefit is derived from multiple chromosomes in natural organisms.

Polyploidy. Polyploidy occurs when a genotype contains homologous chromosomes. A common form is diploidy, in which each chromosome is represented twice in the genotype. Since two alleles are present for each locus, the function D:genotype-->phenotype must include some mapping which reduces two allelic values into one locus value. This is called the dominance map.

By using a dominance map it is possible to carry alleles in the population without testing them in the environment at every generation. This "masking" of alleles may combat the loss of population variance. Diploidy and dominance are examined in Chapter 5.

Circular Chromosomes. Thus far chromosomes have been presented as one-dimensional, linear strings. There are no

precedents in natural systems for multidimensional packages of genetic material, plates of alleles, for example. But there are bacteria and viruses which maintain genetic material in circular form.

On a linear chromosome, the loci at the endpoints are L-1 loci apart, where L is the chromosome length. A locus in the middle of the chromosome, however, is no more than $\frac{L}{2}$ loci distant from any other locus on the chromosome. The loci near the middle will exhibit different patterns of recombination than those on the ends. Their average distance from all other loci will be much smaller than that of an endpoint locus.

The differences in average linkage distances between loci are eliminated by the use of circular chromosomes. The probability of the alleles at any two loci being separated by crossover changes as well. Two alleles on one circular chromosome are separated if a double crossover occurs with one crossover point on either "side" of one allele. In fact, linear chromosomes with single crossover can be viewed as a special case of circular chromosomes with double crossover in which one crossover point is fixed at the end of the chromosome.

It is unknown whether the use of circular chromosomes and double crossover would affect algorithm performance. This work will consider only the simple case of single crossover on linear chromosomes.

2.3.4 Dynamic Genotype Organization

It has been assumed up to now that the representation of the space S is determined initially and remains fixed during the search process. This need not be the case. In nature organism genotypes are rarely static; instead they develop dynamically during adaptation.

There are two main features of the genotype which may be subject to dynamic alteration: the set of loci which are represented and the linkage relationships among those loci. The set of loci can be changed by deleting loci, making additional copies of loci, and by constructing new loci. Linkage is altered by means of the inversion and translocation operators.

Duplication, Deletion, and Random Generation of Loci. There is a convenient representation of solutions to function optimization problems as binary strings. Each binary locus appears once in a genotype and the allelic value of that locus has a straightforward interpretation in the phenotype map D. If the map D is expanded to include a dominance scheme, multiple instances of the same locus can exist in one organism, as is the case in diploid individuals.

However, there is no reasonable interpretation for the absence of a locus in a genotype. The transformation using D could include a random allele generator to expand deficient genotypes into legal phenotypic solutions, but this seems unnatural and of doubtful utility.

Alternatively, the alleles so generated could be fixed as 0's or 1's, but this would be equivalent to the emphasis of a certain subset of S during the search. This could be justified only on the basis of a priori information.

In other problem paradigms, especially those requiring "set" representations (see Section 2.3.5), there is a natural interpretation of multiple and missing loci. Cavicchio (1970) studied duplication and random generation of loci in one such context, but not deletion. He concluded that intrachromosomal duplication was beneficial to the genetic algorithm, but that random generation of new loci was deleterious.

The desirability of allowing multiple and missing loci in genetic algorithms for function optimization has not been investigated. Deletion of loci intuitively seems harmful. If duplication is allowed, some sort of dominance mechanism must accompany it. In this respect, the result of duplication on algorithms for function optimization problems would resemble that of polyploidy except for linkages. A study of diploidy should indicate the benefits of multiple loci in the genotype; further research could then reveal the utility of developing such multiple-loci genotypes dynamically via duplication.

Linkage Alteration: Inversion and Translocation. As mentioned before (Section 2.3.3), the notion of linkage is crucial to a genetic algorithm. Obviously the best genotypic representation for a problem is that which groups

interactive loci together and separates independent loci on separate chromosomes. Of course, if the interactions are unknown, as is usually the case, it is impossible to create such an optimal organization initially.

Inversion and translocation are two operators which alter linkages during offspring creation. Inversion removes a section of a chromosome, flips it end for end, and places it back in the chromosome (see Figure 2.2a). Translocation appends a terminal section of one chromosome to another (Figure 2.2b). Natural systems have mechanisms for both operators; these mechanisms need not be described here.

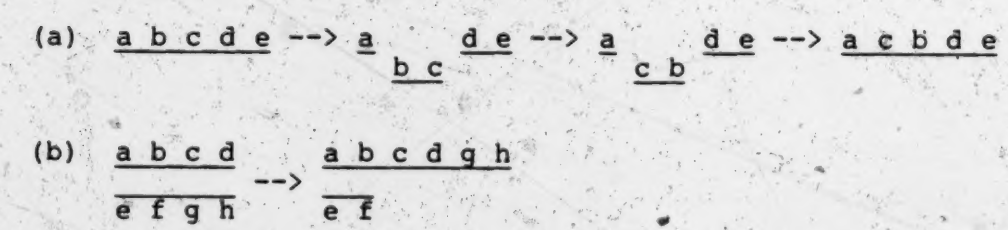


Figure 2.2. Linkage Alteration Operators
(a) Inversion
(b) Translocation

Preliminary investigations (Holland, 1975) indicated that use of inversion would allow a genetic algorithm to search for an optimal genotype organization concurrently with the search for optimal solution points. After experimentation with various implementations, however, Frantz (1972) concluded that inversion did not contribute to algorithm performance. He hypothesized that inversion might be useful in environments requiring longer chromosomes (his

were roughly 25 loci each), or on problems employing a very long period of adaptation. De Jong (1975) agreed that inversion does not play a useful role in genetic algorithms for function optimization problems. One of the biological effects of inversion, which is to prevent crossover between inverted and noninverted chromosomes, has not been examined in research on genetic algorithms.

The translocation operator has not been studied yet within the genetic algorithm. In view of the performance of the inversion operator, there seems to be no good reason to study it extensively. In general, operators on genotypic organization may be too subtle to show any effects when used within genetic algorithms applied to function optimization problems.

2.3.5 The Semantics of Position

The genetic algorithm as presented in this chapter relies on the representation of solutions as chromosomes. Each locus has its own allele set. The allele at one locus plays a specific and well-defined role in the phenotype map D. In a binary representation, one locus is defined to be the "ones" bit, one the "twos" bit, and so on, reflecting the powers of two by which the alleles are to be multiplied. Even if the loci are shuffled on the chromosome, the "meaning" of each locus must be retained. There is semantic significance to each chromosomal position.

Many problems have representations with no intrinsic semantic interpretation for the loci, for instance, tasks in the areas of pattern detection and automatic programming. Assume, for example, that in a pattern detection problem a picture is represented by a 625x625 grid of 0's (light) and 1's (dark). All 625^2 grid sectors on a picture cannot be compared to the sectors in a set of, say, 10 "pattern" pictures in order to locate the closest matching pattern. It is necessary to define a "detector set". Each detector is a function which gives a description of a small set of grid sectors, perhaps 5 sectors. The information from a set of 20 or so detectors is used in comparisons of pictures and patterns, and will permit optimal matching of pattern to picture.

The selection of a detector set for a given pattern set is a difficult problem, possibly suitable for a genetic algorithm (see Cavicchio, 1970). Two representations immediately present themselves. The first is a bit string, a chromosome where each locus i*j has an allele to indicate the presence or absence of grid sector i in detector j. Each locus has a natural meaning, but there are 7,812,500 loci! This may be difficult to implement.

In the second representation, the chromosome consists of a string of grid numbers, each representing one sector which is present in a detector. Each locus has the allele set $\{1,2,\ldots,625^2\}$. Now the chromosome is of length 100; this is manageable. But the loci do not have any

significance other than that of set membership. An allele of 3 in the first locus has the same meaning as the allele 3 appearing in the fifth locus: sector 3 is in the first detector.

A representation such as this without semantics of position may have profound effects on the genetic algorithm. In the set representation above, the crossover operator not only generates duplicate alleles (e.g. a 3 in both locus one and locus five is possible), but it is severely restricted in the manner that it can recombine subsets. A 3 in locus one and a 5 in locus two will stay together in offspring with higher probability than the 3 in locus one and a 7 in locus five, although the 3, 5, and 7 are all members of the same detector and should share the same relationships. Linkage here becomes a liability; it imposes an artificial and undesirable structure on the sets represented.

The theoretical support for genetic algorithms stems mainly from Holland's (1975) analysis of hyperplane (schema) behavior within the Reproductive Plan (see Chapter 3). These hyperplanes are position dependent. Thus the analytical foundation of genetic algorithms may not be valid for problems using set representations. It is unclear whether on a set representation the recombination mechanism of crossover is appropriate, and whether or not the genetic algorithm can be expected to exhibit good performance. Perhaps a new operator (or operators) is needed for the recombination of sets.

Since function optimization problems employ a binary string representation, the discussion of position semantics and set representations need not be pursued further here.

2.4 THE PROBLEM OF LOSS OF POPULATION VARIANCE

Recombination is effective as a search mechanism only as long as there are many different genotypes represented in the population. A population may become homogeneous, that is, all its organisms have similar genotypes. On such a population, crossover ceases to be a tool for exploration. Crossing two identical chromosomes yields the same chromosome again.

Therefore it is desirable to maintain a varied population within the genetic algorithm for the duration of the search process. With respect to function optimization, the loss of population variance is equivalent to convergence of the algorithm (see Martin, 1973, for an analysis of the convergence properties of a simplified Reproductive Plan). If the point of convergence is not optimal, the algorithm has undergone premature convergence.

Premature convergence has been a problem in all implementations of genetic algorithms examined thus far. It stems from the necessary finiteness of the population and an improper balance in the competition for population space between history retention and exploration. An adaptive search algorithm must maintain a history of the search in order to direct further search, and must also provide temporary space for the information resulting from current

evaluations, e.g. the solution just tested and its value. In a genetic algorithm, the population serves both memory functions. A severe imbalance in population usage in favor of history retention results in immediate convergence, probably to a very poor solution. At the other extreme is random search, caused by the ignorance of search history during exploration. Somewhere in between lies the desired balance which yields adaptive search.

The effects of population size, operator rates, parent selection methods, and replacement selection methods on population variance have been studied by Cavicchio (1970) and De Jong (1975). This work continues with a further examination of accurate sampling in parent selection (Chapter 4) and diploidy (Chapter 5) as possible mechanisms for maintaining population variance and improving algorithm performance.

2.5 CONCLUSIONS

There are many possible implementations of the basic Reproductive Plan, and many variations on problem representation. For unconstrained, theoretically unbounded function optimization problems, a binary string representation has been selected, so that the semantics of position are straightforward. Genotypes of single chromosome and two chromosome, diploid organization will be studied.

The goal of the experiments presented in the next three chapters will be to improve algorithm performance on hard functions through reductions in the loss of population variance. Chapter 6 will attempt to verify that a good genetic algorithm is useful in the field of function optimization as a robust problem solving method.

The following particular Reproductive Plan will be employed hereafter:

1. n organisms, $n \geq 50$, are generated according to a uniform random distribution over the set S.
2. 2n parents are selected by sampling from a distribution over the population. This distribution reflects the relative fitness, $\frac{f(s_i)}{\sum_{s \in M} f(s)}$, of each organism s_i.
3. n parental groups are formed by random pairing. Single crossovers occur between chromosomes with a frequency $P_c = 0.6$. Mutation occurs at each locus with a frequency P_m. One offspring is chosen at random from each parental group.
4. The old population is completely replaced by the n offspring.
5. The cycle repeats from step 2.

CHAPTER 3

FUNCTION OPTIMIZATION BY GENETIC ALGORITHMS

The characteristics of functions which are hard to optimize fall into four categories: *a priori* ignorance, static complexity, dynamic ignorance, and dynamic uncertainty (see Section 1.2). In this chapter, many features of static complexity are examined and a set of test functions generated which will contrast the performance of different genetic algorithm implementations. The restriction of attention to features of static complexity should facilitate the development of a genetic algorithm suitable to mathematical function optimization. This algorithm can then be compared to other optimization methods for static, deterministic functions.

3.1 DIFFICULT FUNCTIONS

An optimization algorithm can be described in terms of function features it can handle, features which impair or prevent its operation, and features which do not affect its performance. A brief review of available optimization methods will indicate what some of these features may be. The discussion of function features involves several mathematical terms which are defined in Appendix A. For simplicity, "optimize" will hereafter be interpreted as "maximize". The results are of course applicable to minimization problems as well.

Indirect methods usually require differentiability and/or continuity plus a function in closed form. Thus they are not applicable to functions specified in the form of tables or processes. However, indirect methods can optimize some constrained functions, some multidimensional, unimodal functions, and many multimodal, one dimensional functions.

Random search is blind to most function characteristics. Its expected behavior is determined by the size of the search space, the proportion of good points in the space, and the number of evaluations allowed during search.

Adaptive search methods vary considerably in their ability to cope with variations in constraints, dimensionality, differentiability, and continuity. In general, traditional feedback methods are appropriate to unimodal functions only. Methods in linear, nonlinear, integer and dynamic programming are designed to handle constraints. Dichotomous and Fibonacci search methods are restricted to bounded functions of one dimension, while similar multidimensional tabulation techniques (e.g. multivariate grid search) become unwieldy for large numbers of dimensions.

Some feedback algorithms are designed for unbounded, multidimensional functions. They rely on strict unimodality in the function when depicted in Euclidean space, and thus are susceptible to suboptimal ridges. Gradient methods find the direction of greatest slope and follow it. These

methods require differentiability, although the derivatives can often be approximated using multiple function evaluations. Direct search methods do not use derivatives, but usually exhibit slower convergence properties near the optimum than gradient methods.

Genetic algorithms represent a radical departure from all previous search techniques. It is necessary to determine the extent to which the features of difficulty mentioned above influence or restrict their performance.

3.1.1 Differentiability and Continuity

Genetic algorithms operate on discrete representations of functions and do not use derivatives. As will be seen in the discussion of modality, gross discontinuities in the Euclidean representation of a function need not impair algorithm performance.

3.1.2 Constraints

The ability of genetic algorithms to optimize within constraints has not been studied. The simplest way to incorporate constraints into the environment would be to assign very low values to organisms which fall outside the constraints. It is not clear whether the presence of such "illegal" organisms in the population would hinder or even destroy the search capabilities of the algorithm. This would make an interesting research subject, but will not be investigated here.

3.1.3 Dimensionality

The search performance of genetic algorithms is not affected by the Euclidean dimensionality of the function. Due to the chromosomal structure given to solution points, multidimensional functions have representations similar to one-dimensional functions. As an example, a point in three dimensions is represented by concatenating the loci for each x-value. A function $f(x_1,x_2,x_3)$ defined over the integers $0 \leq x_i \leq 63$, i=1,2,3, will be optimized in exactly the same manner as a one dimensional function g(x), where

$x = 64^2 x_1 + 64x_2 + x_3$,

x_i = remainder of $x/64^{i-1}$ divided by 64 (integer division), and

$g(x) = f(x_1,x_2,x_3)$.

For any bounded function, such a one-dimensional counterpart is easily constructed.

The performance of a genetic algorithm will be influenced not by how many dimensions there are in the domain of a function $f(x_1,\ldots,x_d)$, but by how complex an equivalent one-dimensional function g(x) is.

3.1.4 Modality

Genetic algorithms have the ability to optimize multimodal functions in one or more dimensions (De Jong, 1975), yet some rather simple unimodal functions cause them great difficulty. To understand why, the question of dimensionality must be further examined.

Let $f(x) = 64x - x^2$ be defined over the integers 0 to 63. f is a parabola with one global maximum at 32. A genetic algorithm, when trying to locate this maximum, will very often converge on the value 31, never reaching the "neighboring" point 32. The reason is that the points 31 (011111) and 32 (100000) are not neighboring in the binary representation manipulated by the algorithm. The probability of generating 100000 by crossover or mutation from a population of "good" solutions in the range 20-30 is miniscule, and crossover between points such as 31 and 35 (011111 x 100101) will produce poorer offspring such as 5 (000101) and 63 (111111).

The actions of the crossover and mutation operators are difficult to analyze in terms of one-dimensional Euclidean space. The examination of a one-dimensional counterpart for any function does not increase understanding of how easily it will be optimized by a genetic algorithm. The examination of a multidimensional counterpart does.

Let l_i be the number of loci used to represent x_i, $i=1,\ldots,d$, in a point in d-dimensional Euclidean space, and $L = \sum_{i=1}^{d} l_i$. Consider an L-dimensional discrete space with only two possible values per dimension, 0 and 1. Clearly, all d-dimensional, discrete, bounded functions have many such L-dimensional counterparts.

The distance between two points $(x_1,\ldots,x_d)$ and $(y_1,\ldots,y_d)$ in Euclidean space is $[\sum_{i=1}^{d} (x_i-y_i)^2]^{.5}$. In the

L-dimensional space above, the distance metric is Euclidean distance squared. Since x_i and y_i are limited to 0 and 1,

$$\sum_{i=1}^{L}(x_i-y_i)^2 = \sum_{i=1}^{L}|x_i-y_i| = \sum_{i=1}^{L}(1-\delta_{x_i y_i}),$$

where δ_{ab} (the Kronecker Delta) = 1 if a=b, 0 otherwise.

This particular kind of space is called a Hamming space, and its simplified distance measure is Hamming distance. A genetic algorithm searches the L-dimensional Hamming space counterparts of functions. Mutation changes a point to one of the adjacent points one unit of distance away. When used with low frequency, mutation forms a mechanism for local search about a point.

Crossover is more complicated. Define a k-hyperplane in Hamming space as the set of all points which have the same fixed values for the same k loci. A k-hyperplane can be described by the pattern of fixed loci which all of its elements match. For example, one 2-hyperplane in four dimensional space is {0110,0100,1110,1100}; this set matches the pattern "-1-0".

One approach to searching Hamming or other spaces is to determine which features, or combinations of features (hyperplanes), are responsible for good solutions, and to recombine sets of good features to yield still better solutions. Although the apportionment of credit to hyperplanes would be a valuable technique for search, it cannot be implemented explicitly.

In L-dimensional Hamming space there are 3^L-1 hyperplanes of interest (the 0-hyperplane does not figure in apportionment of credit). In order to identify good hyperplanes, it is necessary to sample points from many hyperplanes. Each point in the space is a member of 2^L hyperplanes, however, so evaluating one point yields information about many hyperplanes. Thus it is possible to sample sufficient points to indicate the utility of most hyperplanes, but storage of the summary information about each hyperplane is difficult.

Holland (1975) has shown that in a genetic algorithm the population M can be expected to contain each hyperplane with frequency proportional to its observed utility. The population provides <u>implicit</u> storage of hyperplane utilities. A genetic algorithm, therefore, has the capability to search by means of apportionment of credit, provided that some mechanism for the recombination of hyperplanes is employed. This is the purpose of crossover.

Crossover recombines hyperplanes in the population. In particular, the crossover operator, described in Section 2.2.3, will keep closely linked loci together while separating distant loci to effect recombination. As an illustration, consider a four dimensional space. An instance of the 2-hyperplane 10-- may occasionally be broken by crossover between loci one and two, but more often crossover will occur between loci two and four, to recombine it with hyperplanes such as --11 and ---0. An instance of

the hyperplane 1--0, on the other hand, will usually be destroyed by crossover, although it may be regenerated in the offspring from a similar hyperplane on the second chromosome.

The operation of the genetic algorithm does not preclude the evaluation of hyperplanes containing distantly linked loci. However, more recombination and evaluation will occur for hyperplanes with closely linked loci. In general, too, more evaluations will occur for hyperplanes with few defining loci than for those with many.

A genetic algorithm will be more successful in apportioning credit for performance to hyperplanes with a few, closely linked loci than to hyperplanes with many or distantly linked loci. If good values for a function can be attributed to the independent contributions of each locus, or of small sets of loci, then the genetic algorithm should be able to combine the independent effects to produce an optimal solution. If credit cannot be allocated independently to small, close groups of loci, then there must be dependencies, or nonlinearities, between large numbers of loci or distantly linked loci. These nonlinearities could impair algorithm performance.

A genetic algorithm should be effective optimizing functions which are linearly independent over the loci in Hamming space. In terms of the Euclidean space over which the function may originally be defined, bounds are necessary, differentiability and dimensionality are

unimportant, and discontinuities and multimodality do not restrict the algorithm, but must be viewed in light of their effects on the nonlinearity of the function in Hamming space.

Since multimodality has been the bane of most traditional optimization methods, the ability of genetic algorithms to optimize multimodal functions may prove to be their most important feature. This work will concentrate on the development of genetic algorithms for multimodal functions, using test functions which exhibit various locus dependencies in their Hamming space representations. All functions will be unconstrained, but bounded. Most will be one dimensional in their original representation, for ease of presentation.

3.2 DESIGN OF EXPERIMENTS

The experiments to test features of and to compare genetic algorithms to other methods consisted of two or more algorithms run on a set of test functions. In order to identify interactions between algorithm types and functions, a factorial design was used, for all experiments, with analysis of variance for main effects and their interactions.

F-ratios were defined as the mean square of effects over the mean square of an error term, or

$$F(A \times B) = \frac{MS(A \times B)}{MS(error)}.$$

Some of the experiments involved repeated measures (all factors listed after the replication factor in the

experiment design). For interactions involving repeated measures, the denominator of the F-ratio was MS(within cell x any repeated measures); otherwise it was simply MS(within cell) (Winer, 1962). The level of statistical significance of effects was measured at 5%, with sufficient numbers of runs to satisfy assumptions of normality in the analysis (at least 30 degrees of freedom for the F-ratio denominator).

Experiments were run on a PDP11/60 under the UNIX operating system using simulation programs written in the language "C". Analytical results were obtained using programs from the IMSL (1975) statistical package running on an Amdahl470/V7 under the Michigan Terminal System.

Genetic algorithms are stochastic processes. The validity of experiments involving these algorithms is dependent upon the degree of randomness of the pseudo-random numbers employed. One run of a genetic algorithm with diploid population of size 200, chromosomes of length 50, and duration 100 generations could require as many as 40,000 (about 2^{15}) random numbers for mutation alone. The UNIX random number generator, which uses integer arithmetic and has a cycle period of $2^{15}-1$, was considered inadequate for experiments of several runs.

The multiplicative congruential generator

$$x_{i+1} = 16384 * x_i \pmod{268435399}$$

was designed according to the mathematical requisites of Downham and Roberts (1967), to provide a large period ($2^{28}-58$) by using "C" double-precision floating point

arithmetic. Sequences of numbers from this generator passed several statistical tests of randomness at the 5% level, including the X^2 uniformity, d^2, "runs up and down", and serial tests (Downham and Roberts, 1967).

For simplicity, genetic algorithms will be identified in experiment summaries only by those features which differentiate them from the algorithm finally chosen by De Jong (1975). The parameters of this default implementation are listed below. Many of these will not be fully explained until later chapters.

representation = binary
population size (n) = 50
generation size (g) = 50
number of chromosomes per haploid genotype = 1
ploidy = 1 (haploidy)
dominance scheme = random
crossover probability per chromosome (P_c) = 0.6
number of crossover points per chromosome = 1
mutation probability per locus (P_m) = 0.001
parent selection = stochastic without replacement
parent distribution base = worst all-time
mating = random
replacement selection = random

3.3 MEASURES OF PERFORMANCE

All optimization methods were run in fixed resource mode, terminating on a preset number of function evaluations. Comparisons are on the basis of overall best

performance, or the maximum function value achieved on any evaluation up to the termination point, because this seems to be the most important measure for function optimization. Online performance was also monitored as an aid to understanding algorithm strengths and weaknesses. Offline performance was not analyzed, since plots of overall best performance over time seemed to convey sufficient information on the rate of improvement of overall best values.

For those experiments concerned with changes in population variance, it was necessary to formulate possible measures of variance. Let a population contain chromosomes of length L, with B alleles (for base, e.g. binary, decimal) possible at each locus. For each allele a_{ij} at locus i, i=1,...,L, j=1,...,B, let h_{ij} be the proportion of the chromosomes in the current population which contain a_{ij}.

Fixation of loci is measured as the proportion of loci which are homogeneous for one allele, that is, for which the population has converged completely.

$$\text{Fixation} = \frac{1}{L}\sum_{i=1}^{L}\sum_{j=1}^{B}\lfloor h_{ij} \rfloor .$$

Obviously $\sum_{j=1}^{B} h_{ij} = 1$ for i=1,...,L. Therefore the average frequency of alleles $\bar{h} = \frac{1}{LB}\sum_{i=1}^{L}\sum_{j=1}^{B} h_{ij}$ is simply $\frac{1}{B}$.

The sample variance of the allele frequencies is computed as

$$\text{Variance of Frequency} = \frac{1}{LB-1}\sum_{i=1}^{L}\sum_{j=1}^{B}(h_{ij} - \bar{h})^2$$

$$= \frac{1}{LB-1}[(\sum_{i=1}^{L}\sum_{j=1}^{B}h_{ij}^2) - \frac{L}{B}].$$

Fixation indicates how many loci have reached extremes of allele frequency, while variance of frequency indicates the amount of variation of all alleles from the frequency value associated with high population variance. It may also be useful to detect the degree to which one allele is dominating a locus, in other words, to measure the amount of convergence at each locus. For this purpose, the average maximum frequency is defined as

$$\text{Average Maximum Frequency} = \frac{1}{L}\sum_{i=1}^{L}\max_{j=1,B}\{h_{ij}\}.$$

To determine whether all loci are converging similarly, the sample variance of maximum frequencies may be used:

$$\text{Variance of Maximum Frequency} = \frac{1}{L-1}\sum_{i=1}^{L}(\max_{j=1,B}\{h_{ij}\} - \text{AMF})^2,$$

where AMF is the Average Maximum Frequency.

3.4 TEST FUNCTION SET I

The set of test functions designed for Chapters 4 and 5 are defined over 2^{30} points with one global maximum of value 100 (or nearly 100). The set includes functions which in Hamming space are linearly independent, and functions which have closely linked and distantly linked dependent loci.

3.4.1 Function 1

Define $H = (h_1,\ldots,h_{30})$ as the point $(1,0,0,\ldots,0)$; $h_1=1$ and $h_i=0$ for $i=2,\ldots,30$. Let $X = (x_1,\ldots,x_{30})$ be a point in 30-dimensional Hamming space.

$$f1(X) = -80 + 6\sum_{i=1}^{30} \delta_{x_i h_i} ,$$

where $\delta_{ab} = 1$ if $a=b$, 0 otherwise.

Maximum: $f1(1,0,0,\ldots,0) = 100$

Euclidean space modality: an equivalent one dimensional function, fle, over the integers $[0,2^{30}-1]$ would have 2^{29} (5×10^8) peaks.

f1 is linearly independent over the loci in Hamming space. It should be very easy for a genetic algorithm to optimize. The equivalent one dimensional function fle is shown in Figure 3.1.

3.4.2 Function 2

Let $X = (x_1,\ldots,x_{30})$ be a point in 30-dimensional Hamming space. Group the terms x_i as follows:

$$G_1 = \{x_1,x_2,x_3,x_{29},x_{30}\}$$
$$G_2 = \{x_4,x_5,x_{26},x_{27},x_{28}\}$$
$$G_3 = \{x_6,x_{11},x_{18},x_{22},x_{25}\}$$
$$G_4 = \{x_7,x_{14},x_{17},x_{20},x_{24}\}$$
$$G_5 = \{x_9,x_{10},x_{15},x_{21},x_{23}\}$$
$$G_6 = \{x_8,x_{12},x_{13},x_{16},x_{19}\}$$

Define the value V of each group for a given X on the basis of the number of 1's occurring in the group. For $i=1,\ldots,6$,

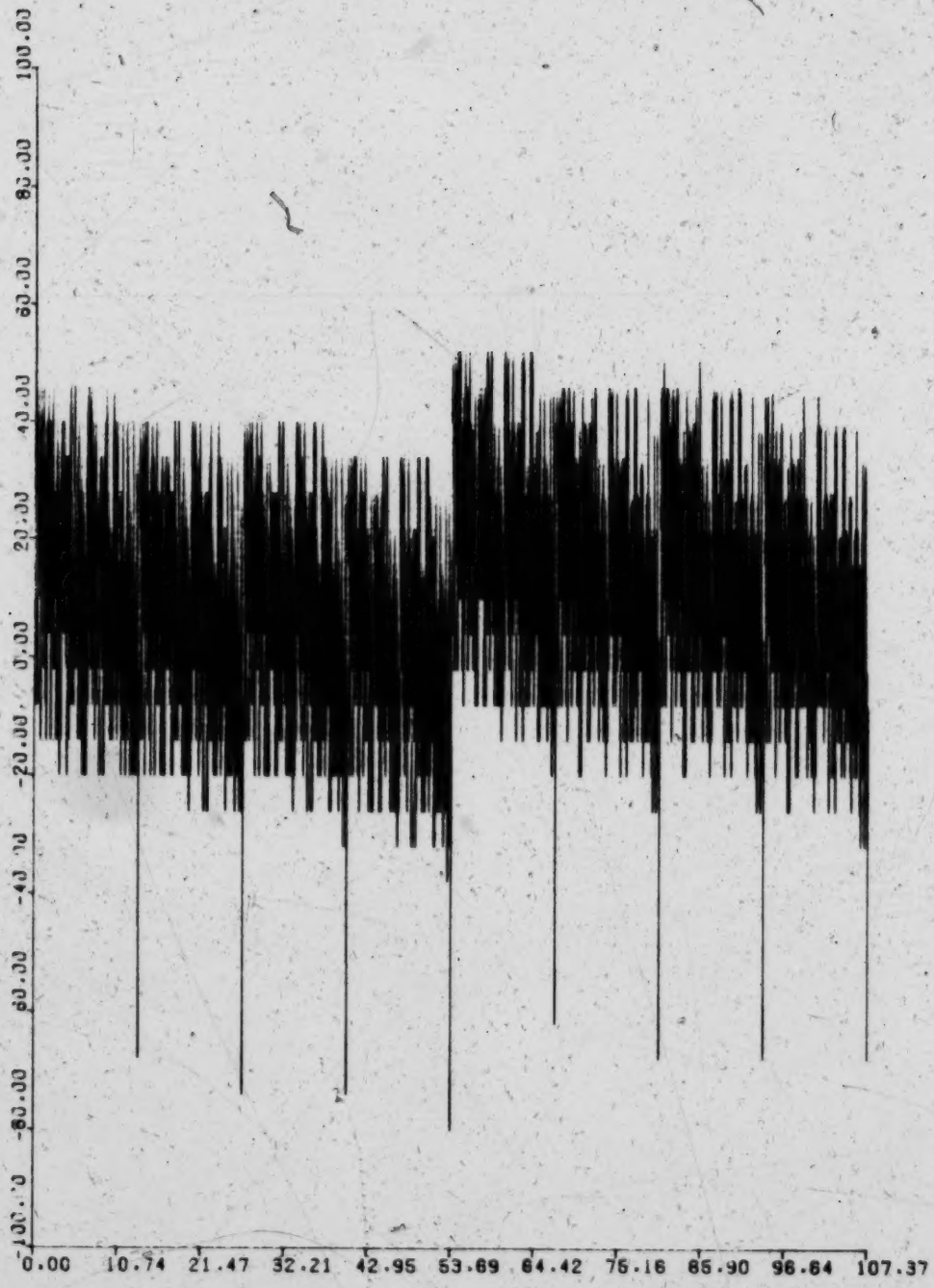


Figure 3.1. Function fle.
for $0 \leq x \leq 1073741823$
(5000 points plotted)

number of 1's in G_i: 5 4 3 2 1 0

value V_i: 30 15 0 5 10 20

Then

$$f2(X) = -80 + \sum_{i=1}^{6} V_i.$$

Maximum: f(1,1,...,1) = 100

Euclidean space modality: An equivalent one dimensional function, f2e, over the integers $[0,2^{30}-1]$ would be highly multimodal.

f2 should be very difficult for a genetic algorithm to optimize. The loci in each group are dependent and located all over the chromosome. The equivalent function f2e is shown in Figure 3.2.

3.4.3 Function 3

The next four functions are Fourier sums defined on one dimension for the integers $[0,2^{30}-1]$.

$$f3(X) = \sum_{i=23}^{29} c_i \sin(\pi(X/2^i+1))$$

for c_i = 31.41177, 31.41177, 31.41177, 15.7058853, 15.7058853, 15.7058853, 15.7058853, i=23,...,29.

Maximum: f3(1001230000) = 100.0

Euclidean space modality: 2^6 (64) peaks.

Each sine wave in the sum for f3 is constructed with period 2^{i+1} and first peak at $3 \cdot 2^{i-1}$, so that if locus 30-i is 1 the evaluation for that sine wave will be superior to when locus 30-i is 0. Thus in Hamming space f3 will have seven important, but independent, loci grouped at one end of

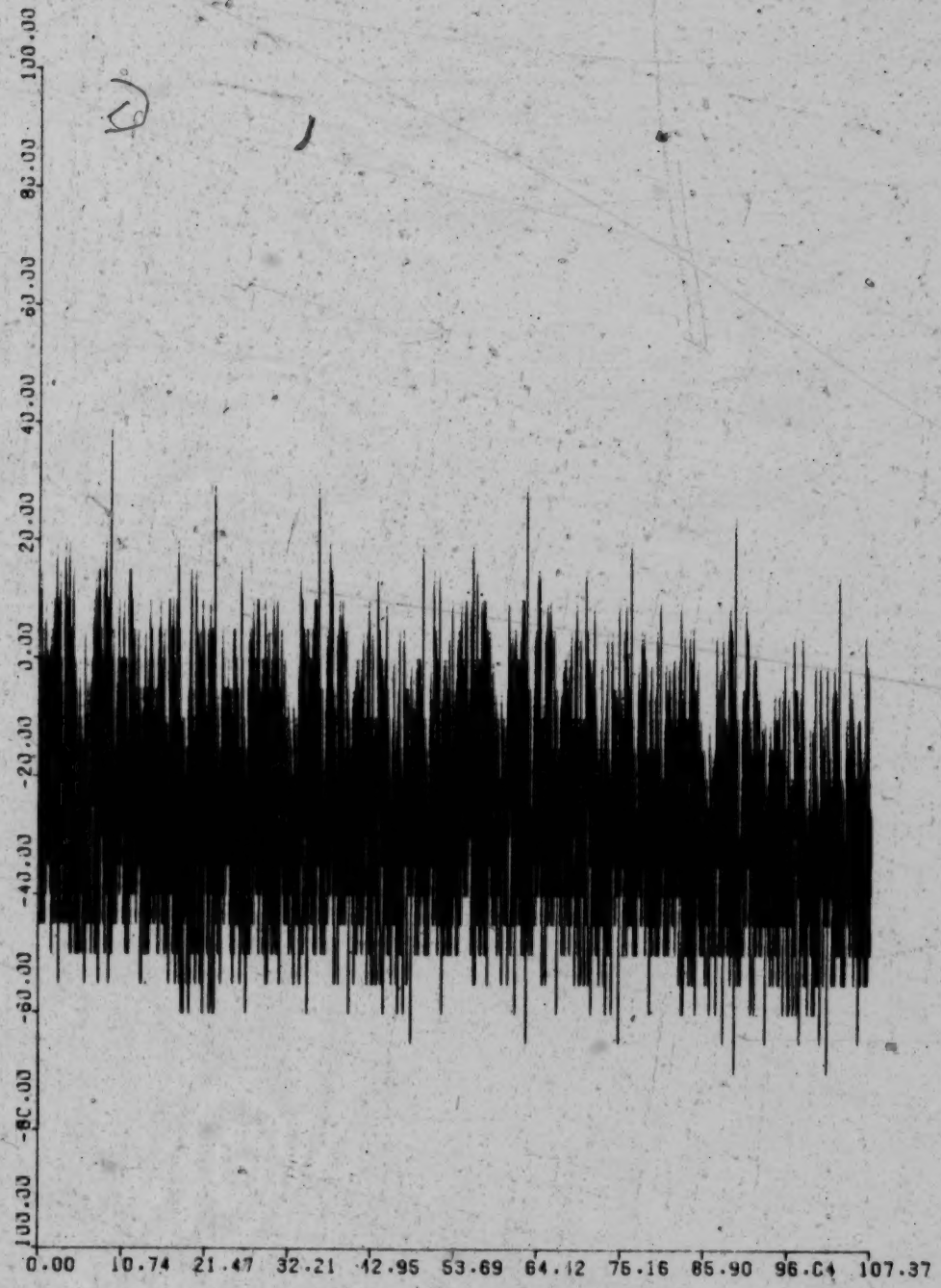


Figure 3.2. Function f2e.
for $0 \leq x \leq 1073741823$
(5000 points plotted)

the chromosome. f3 is shown in Figure 3.3.

3.4.4 Function 4

$$f4(X) = \sum c_i \sin(\pi(X/2^i+1))$$

for i=1,5,10,15,20,25,29 and c_i = 20.05259, 20.05259, 20.05259, 10.026295, 10.026295, 10.026295, 10.026295.

Maximum: f4(788053000) = 100.0

Euclidean space modality: 2^{29} (5×10^8) peaks.

f4 is similar to f3, except that the important loci are widely distributed on the chromosome. f4 is shown in Figure 3.4.

3.4.5 Function 5

$$f5(X) = \sum_{i=1}^{7} c_i \sin(2\pi X/b_i)$$

for c_i = 26343867, 26343867, 26343867, 13.17193, 13.17193, 13.17193, 13.17193 and $b_i = 53^4, 11^7, 19^6, 13^7, 41^5, 5^{12}, 17^7$, i=1,...,7.

Maximum: f5(1061130000) = 100.0

Euclidean space modality: approximately 68 peaks.

The periods of the sine waves in f5 are odd, relatively prime, and in the range 2^{24} to 2^{30}. This means that there will be a set of important loci which are dependent and closely linked at one end of the chromosome. f5 is shown in Figure 3.5.

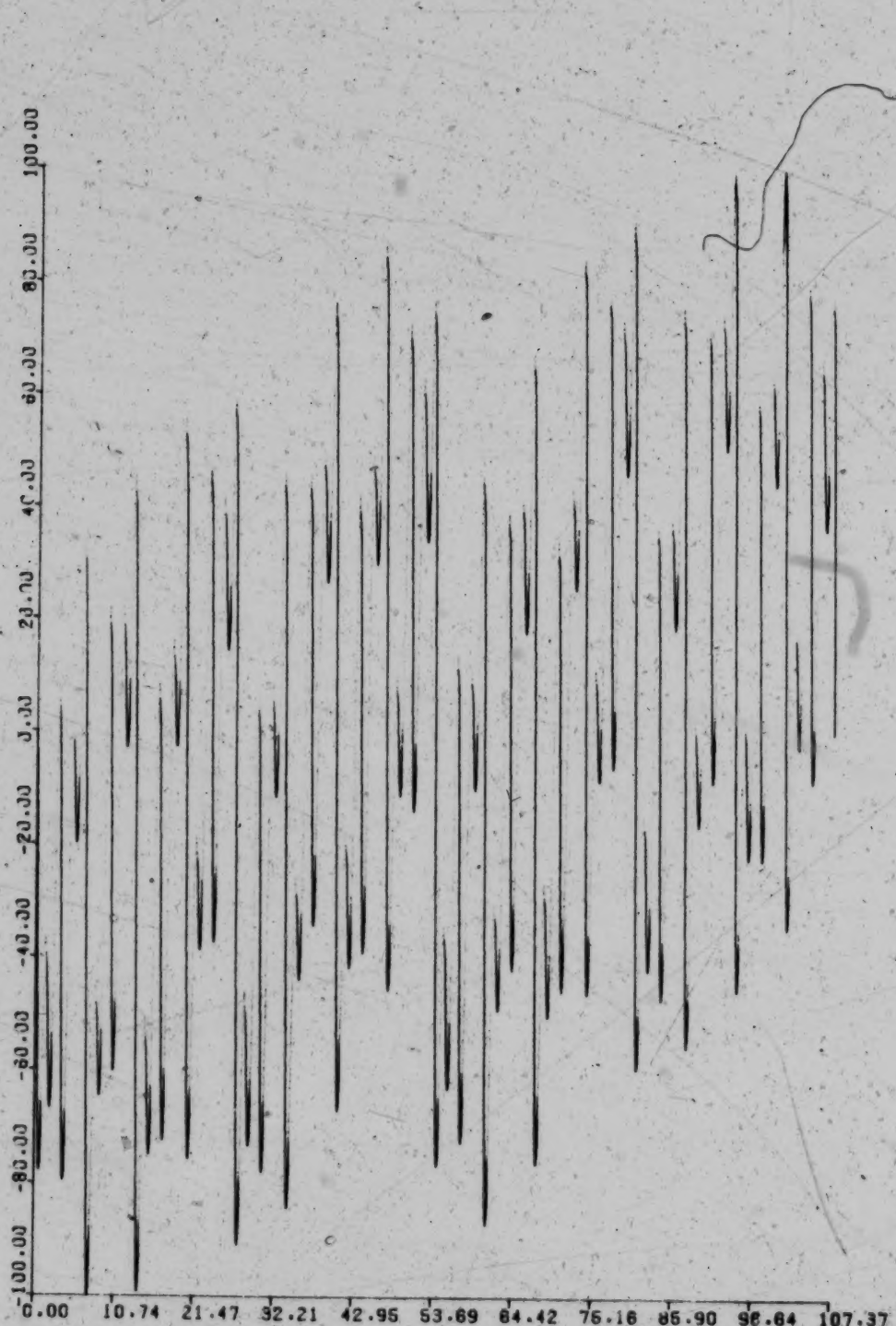


Figure 3.3. Function f3.
for $0 \leq x \leq 1073741823$
(5000 points plotted)

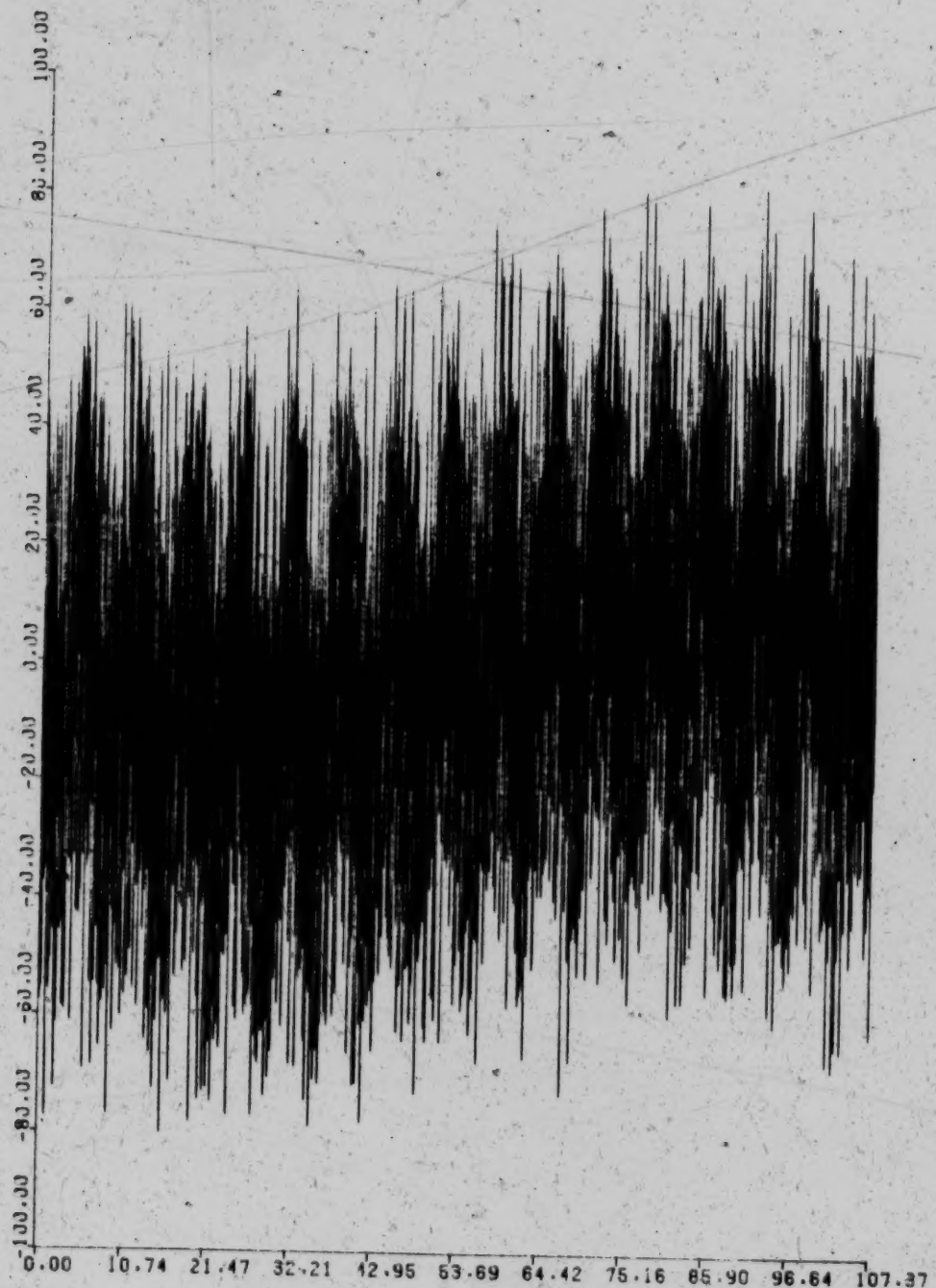


Figure 3.4. Function f4.
for $0 \leq x \leq 1073741823$
(5000 points plotted)

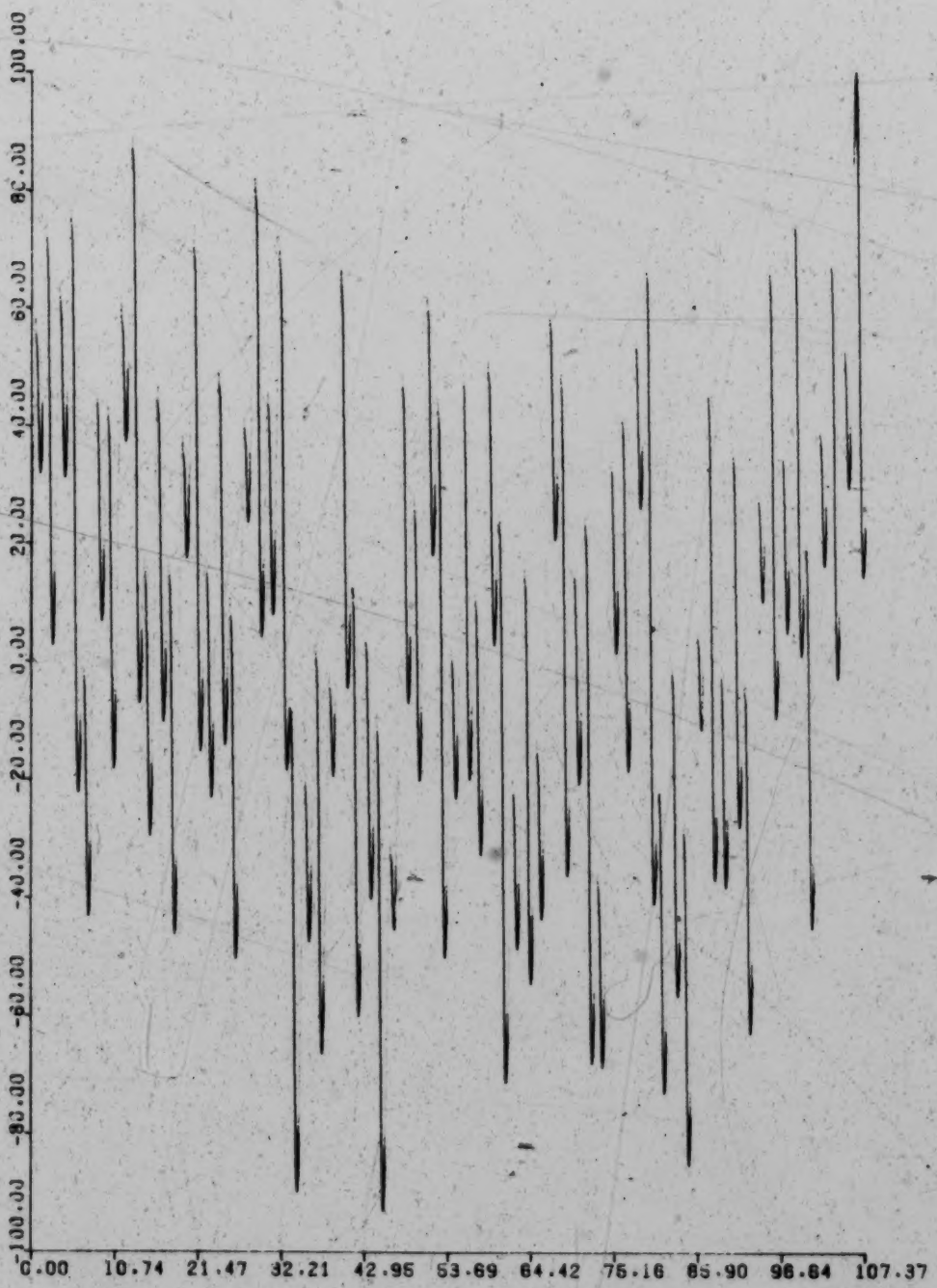


Figure 3.5. Function f5.
for $0 \leq x \leq 1073741823$
(5000 points plotted)

3.4.6 Function 6

$$f6(X) = \sum_{i=1}^{7} c_i \sin(2\pi X/b_i)$$

for c_i = 20.4052, 20.4052, 20.4052, 10.20291, 10.20291, 10.20291, 10.20291 and b_i = 5, 31, 13^3, 7^5, 43^4, 53^4, 17^7, i=1,...,7.

Maximum: f6(240780000) = 100.0

Euclidean space modality: approximately $2^{29}/5$ (1.1×10^8) peaks.

As in f5, the sine periods are odd and relatively prime. The wide range of period magnitudes should cause dependencies between loci distributed over the length of the chromosome. f6 is shown in Figure 3.6.

3.4.7 Function 7

Let $Y = X/2^{30}$ for any integer X in the range $[0, 2^{30}-1]$.

$$f7(X) = 100\ (1-Y)^4 \sin(16\pi(Y+.05874169672)^{-.25}).$$

Maximum: f(0) = 99.999985

Euclidean space modality: 8 peaks.

Function 7, a dampened sinusoid, is included because of its difficulty for all optimization methods. Its lack of a regular period should impair the performance of a genetic algorithm. f7 is shown is Figure 3.7.

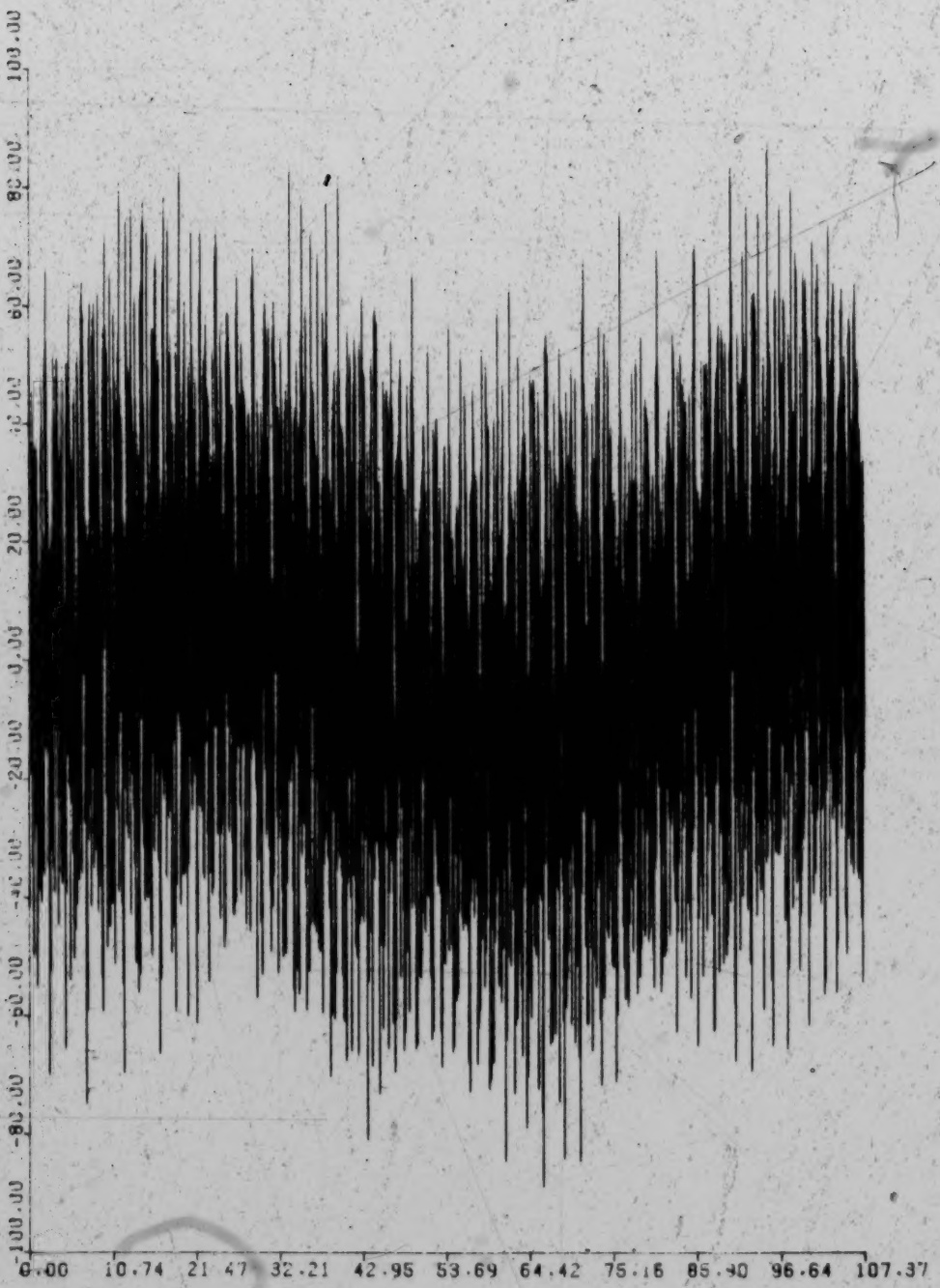


Figure 3.6. Function f6.
for $0 \leq x \leq 1073741823$
(5000 points plotted)

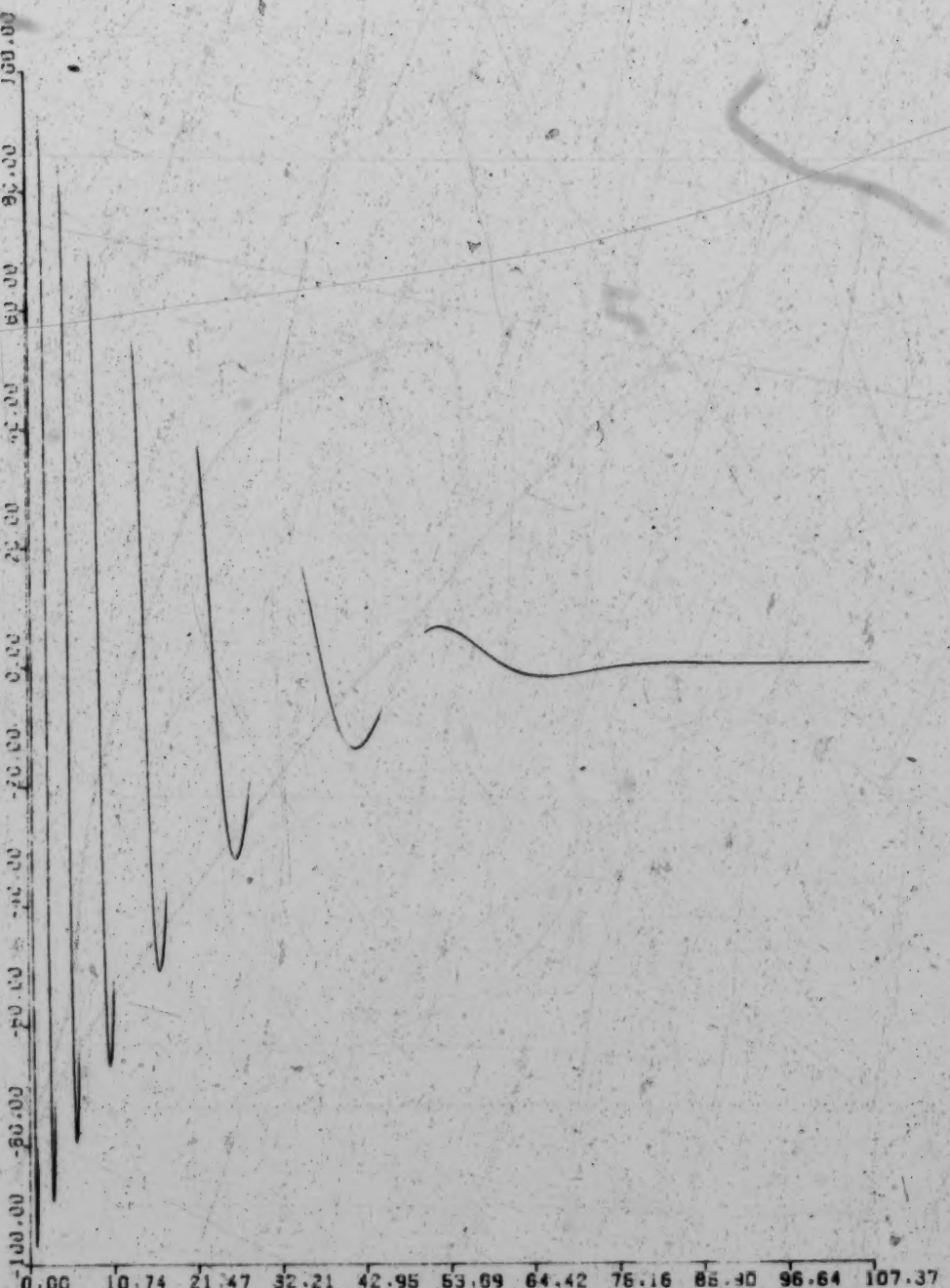


Figure 3.7. Function f7.
for $0 \leq x \leq 1073741823$
(5000 points plotted)

3.5 EXPERIMENT 1: RANDOM SEARCH VS. THE DEFAULT ALGORITHM

The first experiment was designed to evaluate Test Function Set I, evaluate the various measures of population variance, and determine an appropriate run length for later experiments. The default genetic algorithm and random search were run on all seven functions. Factors in the experiment were:

1. Functions (1, 2, 3, 4, 5, 6, 7)
2. Replications (10)
3. Algorithm (random search,
   default genetic algorithm)

The results are listed in Table 3.1.

As expected, the genetic algorithm performed well on Functions 1, 3, and 4, which are linearly independent in Hamming space. The result for Function 5 indicates that locus dependencies between seven loci hamper a genetic algorithm somewhat, even though the dependent loci are closely linked. Functions 2, 5, 6, and 7 are difficult functions for a genetic algorithm; in fact, on the last three functions random search was superior.

The genetic algorithm performed well on both Functions 3 and 4, indicating that as long as the loci are independent, it makes no difference which loci on the chromosome are important in the function evaluation. Since both algorithms reached near-optimal points on Function 3, this function will be removed from Test Function Set I. Functions 1 and 4 will be retained as benchmark "easy"

functions, and Functions 2, 5, 6, and 7 as functions on which algorithm performance can be improved.

The variance measures for the genetic algorithm population on each of the functions are given in Table 3.2. Variance of Frequency parallels Average Maximum Frequency in indicating overall population variance, but displays smaller differentials. Variance of Maximum Frequency exhibits a negative correlation to Average Maximum Frequency. This should be true whenever the Average Maximum Frequency approaches one. Average Maximum Frequency and Fixation seem to provide the clearest measures of population variance.

The overall best performance of both algorithms is plotted against time in the graphs of Figures 3.8a through 3.8c. In all cases except Function 7, the genetic algorithm had located its best value after 5000 to 7500 function evaluations. On Function 7 the algorithm reached a plateau at about 8500 evaluations.

The Average Maximum Frequency of the genetic algorithm is plotted against time in Figure 3.9. On each of the functions, the algorithm reached a stable population of small variance after 6500 to 7000 evaluations. Search from then on would have been limited to random search by mutation. This explains the plateaus observed in the graphs of overall best performance.

Further experiments will be terminated after 8000 function evaluations to allow the genetic algorithm its full period of search by recombination.

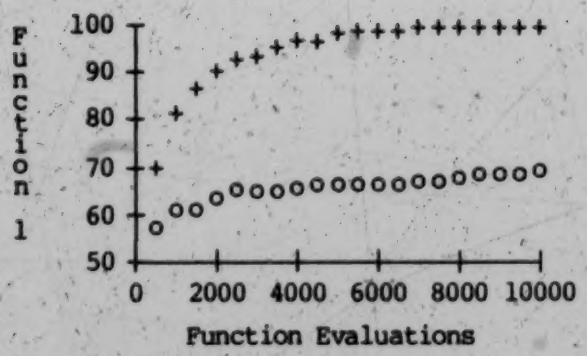
Table 3.1. Experiment 1: Random Search vs. Default Algorithm.
Overall Best Performance after 10000 Function Evaluations
(results averaged over 10 runs)

Algorithm:	Random Search	Default Genetic Algorithm
Function:		
1	69.400	99.400
2	37.500	57.000
3	99.987	99.466
4	92.561	98.763
5	99.988	90.194
6	92.962	89.016
7	99.860	91.655

Source of Variation	M.S.	D.F.	F-ratio	Significant
Function	6606.00	6	245.0	*
Algorithm	789.00	1	24.9	*
Function x Algorithm	1117.00	6	35.2	*
Within Cell	27.01	63		
Algorithm x Within Cell	31.71	63		

Table 3.2. Experiment 1: Random Search vs. Default Algorithm.
Population Variance for Default Algorithm
Measured after 10000 Function Evaluations
(results averaged over 10 runs)

Measure:	Variance of Frequency	Proportion of Fixation	Average Maximum Frequency	Variance of Maximum Frequency
Function:				
1	.229	.677	.969	.0050
2	.230	.650	.970	.0049
3	.196	.537	.918	.0180
4	.212	.593	.941	.0142
5	.194	.513	.914	.0192
6	.247	.840	.991	.0016
7	.200	.570	.922	.0180



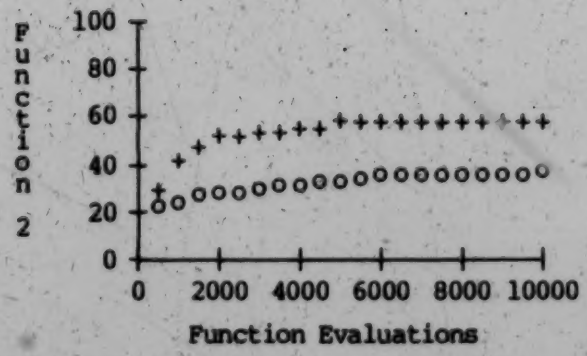
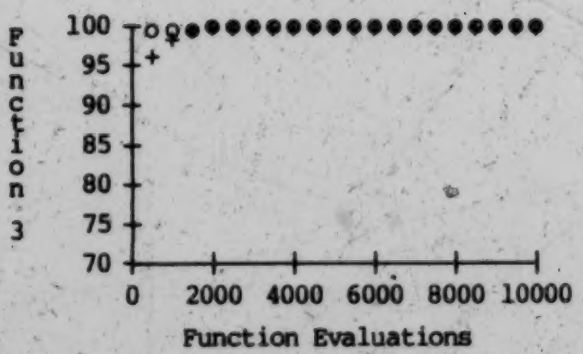


Figure 3.8a. Experiment 1: Random Search vs. Default Algorithm.
Overall Best Performance on Functions 1 and 2
(o) Random Search
(+) Default Genetic Algorithm



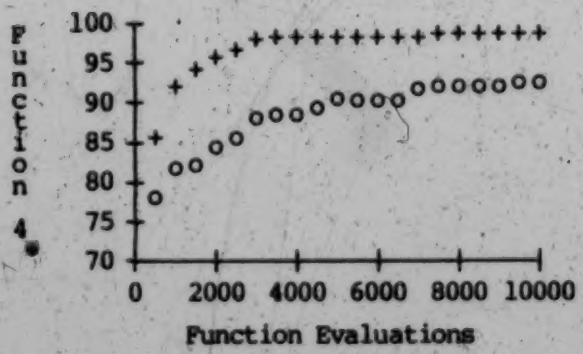
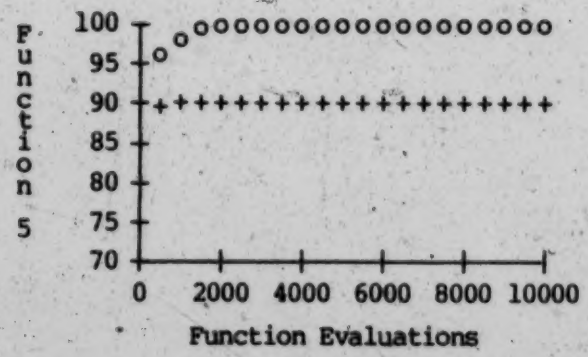
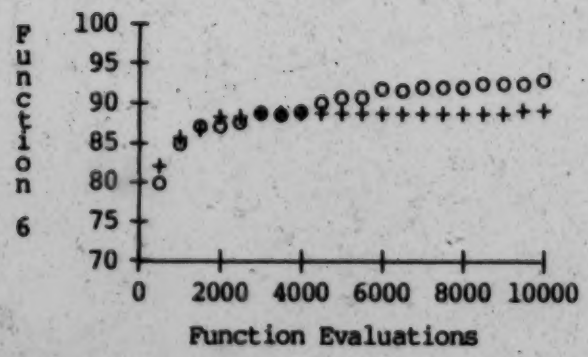


Figure 3.8b. Experiment 1: Random Search vs. Default Algorithm.
Overall Best Performance on Functions 3 and 4
(o) Random Search
(+) Default Genetic Algorithm





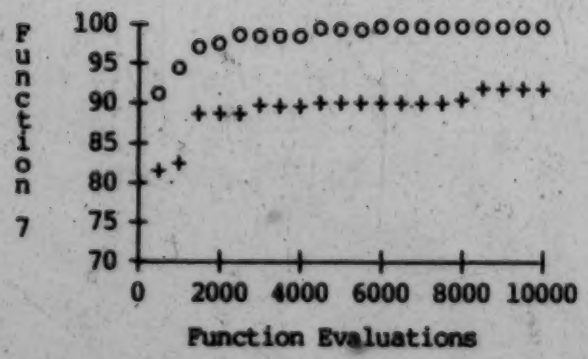
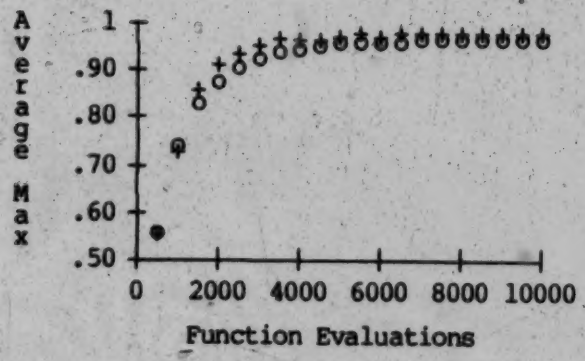
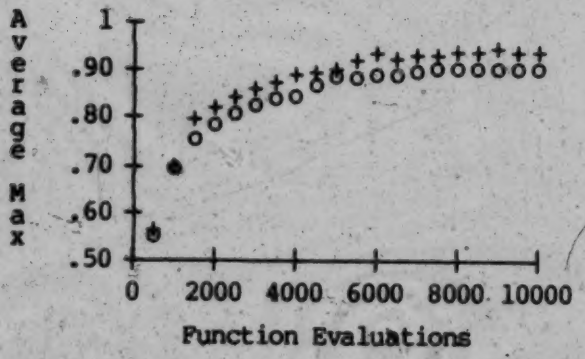


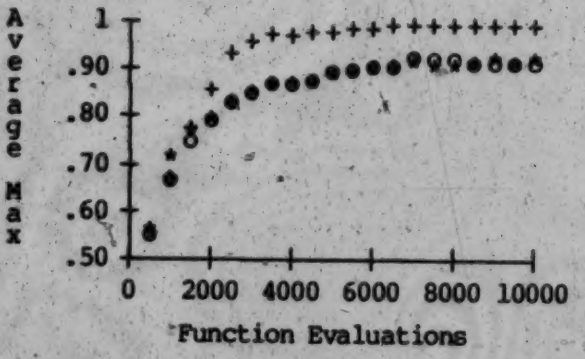
Figure 3.8c. Experiment 1: Random Search vs. Default Algorithm.
Overall Best Performance on Functions 5, 6, and 7
(o) Random Search
(+) Default Genetic Algorithm



a. (o) Function 1
(+) Function 2



b. (o) Function 3
(+) Function 4



c. (o) Function 5
(+) Function 6
(*) Function 7

Figure 3.9. Experiment 1: Random Search vs. Default Algorithm. Average Maximum Frequency for Default Algorithm

3.6 EXPERIMENT 2: POPULATION SIZE AND MUTATION RATE

De Jong (1975) constructed several experiments to evaluate the parameters of population size, mutation rate, and crossover rate. His performance measures, however, were online and offline performance, with an emphasis on online. For superior online performance, the random element of mutation was minimized. Experiment 2 was performed to test his results for mutation rates under the measure of overall best performance. Since levels of mutation needed to maintain population variance are closely tied to population size, population size was included as a factor in the experiment.

Genetic algorithms with populations of 50, 100, and 200 were run on Test Function Set I. Three mutation rates were used: 0.001 (De Jong's optimal value), which is $\frac{1}{5n}$ for n=200, 0.01, which is $\frac{1}{n}$ for n=100, and 0.02, which is $\frac{1}{n}$ for n=50. The experiment factors were:

1. Function (1, 2, 4, 5, 6, 7)
2. Population Size (50, 100, 200)
3. Replications (4)
4. Mutation Rate (0.001, 0.01, 0.02)

The results are given in Table 3.3.

The two factor interactions listed are easy to understand. The interaction between population size and mutation rate supports the theory that the mutation rate needed to maintain allelic variance is inversely proportional to the population size (Holland, 1975).

Table 3.3. Experiment 2: Population Size and Mutation Rate. Overall Best Performance after 8000 Function Evaluations (results averaged over 4 runs)

Function	Population Size	Mutation Rate .001	.01	.02
1	50	100.000	100.000	92.500
	100	100.000	98.500	97.000
	200	100.000	100.000	91.000
2	50	60.000	61.250	66.250
	100	60.000	62.500	65.000
	200	67.500	66.250	65.000
4	50	98.538	98.368	98.678
	100	98.844	98.333	98.306
	200	98.841	98.653	97.186
5	50	89.079	96.432	91.885
	100	99.999	99.990	99.998
	200	99.998	99.445	99.993
6	50	89.261	92.736	95.043
	100	92.433	94.481	93.873
	200	96.420	95.122	95.534
7	50	90.504	99.993	99.960
	100	92.180	98.391	99.910
	200	98.093	96.404	99.966

Source of Variation	M.S.	D.F.	F-ratio	Significant
Function	6695.00	5	305.00	*
Population Size	115.50	2	5.26	*
Mutation Rate	35.75	2	2.21	
Function x Pop. Size	42.16	10	1.92	
Function x Mut. Rate	61.83	10	3.82	*
Pop. Size x Mut. Rate	47.81	4	2.96	*
Function x Pop. Size x Mut. Rate	12.52	20	.77	
Within Cell	21.96	54		
Mut. Rate x Within Cell	16.17	108		

Function and mutation rate also show a significant interaction. A high mutation rate improved performance on the difficult functions, while degrading performance on the easy functions. Introducing more mutations pushed the behavior of the genetic algorithm towards that of random search. Performance on the damped sinusoid (Function 7) especially was improved, but it is clear that mutation at a rate of 0.02 or higher is generally undesirable.

The significance of population size was unexpected. A population of 200 was superior to those of 100 and 50 in most cases. This may be attributed to the increased population variance which derives from large population size. Table 3.4 shows the significant decrease in Average Maximum Frequency caused by increase in population size. Although high mutation rates also increase variance, they do so at the expense of the adaptive features of the algorithm. Apparently this deleterious effect cancels the improvement in performance of the corresponding increase in variance.

The significance of population size may also be the result of variation between runs in the experiment. As population size increases, the population variance increases, but the expected variance between replications at one population size decreases. This effect may prejudice the analysis of variance somewhat.

Table 3.5 charts the fixation behavior in Experiment 2. For 100 or more chromosomes in the population, fixation is rare to nonexistent. Although great differences in variance

Table 3.4. Experiment 2: Population Size and Mutation Rate. Average Maximum Frequency after 8000 Function Evaluations (results averaged over 4 runs)

Function	Population Size	Mutation Rate		
		.001	.01	.02
1	50	.969	.870	.775
	100	.944	.831	.748
	200	.888	.819	.750
2	50	.979	.865	.791
	100	.960	.856	.785
	200	.872	.783	.755
4	50	.945	.790	.720
	100	.855	.733	.705
	200	.752	.704	.668
5	50	.914	.785	.712
	100	.850	.739	.686
	200	.719	.663	.648
6	50	.991	.825	.680
	100	.945	.727	.632
	200	.741	.665	.595
7	50	.927	.810	.736
	100	.830	.738	.688
	200	.760	.698	.663

Source of Variation	M.S.	D.F.	F-ratio	Significant
Function	.07929	5	70.40	*
Population Size	.21000	2	186.00	*
Mutation Rate	.54680	2	778.00	*
Function x Pop. Size	.00570	10	5.06	*
Function x Mut. Rate	.00581	10	8.27	*
Pop. Size x Mut. Rate	.02028	4	28.90	*
Function x Pop. Size x Mut. Rate	.00110	20	1.56	
Within Cell	.00126	54		
Mut. Rate x Within Cell	.00070	108		

Table 3.5. Experiment 2: Population Size and Mutation Rate.
Fixation after 8000 Function Evaluations
(results averaged over 4 runs)

Function	Population Size	Mutation Rate		
		.001	.01	.02
1	50	.617	.050	.0
	100	.408	.008	.0
	200	.042	.0	.0
2	50	.792	.075	.0
	100	.442	.0	.0
	200	.100	.0	.0
4	50	.600	.067	.0
	100	.233	.0	.0
	200	.058	.0	.0
5	50	.525	.067	.008
	100	.275	.025	.0
	200	.033	.008	.0
6	50	.775	.067	.0
	100	.467	.0	.0
	200	.008	.0	.0
7	50	.517	.083	.017
	100	.267	.025	.0
	200	.075	.0	.0

Source of Variation	M.S.	D.F.	F-ratio	Significant
Function	.01764	5	10.00	*
Population Size	.00861	2	490.00	*
Mutation Rate	.02663	2	1410.00	*
Function x Pop. Size	.00678	10	3.86	*
Function x Mut. Rate	.02196	10	11.60	*
Pop. Size x Mut. Rate	.61120	4	323.00	*
Function x Pop. Size x Mut. Rate	.00727	20	3.85	*
Within Cell	.00176	54		
Mut. Rate x Within Cell	.00189	108		

still exist in these populations, such differences are not reflected by this measure.

For overall best performance, a population size of 200 and mutation rate of 0.001 are preferable, while a population of 50 and mutation rate of 0.001 are clearly inferior. This in no way contradicts De Jong's findings, as the online values for Experiment 2 show (Table 3.6).

In Figure 3.10, the overall best values for the algorithm with population 200 and mutation rate 0.001 are plotted. It appears that with the possible exception of Function 6, the algorithm has reached a plateau by 8000 function evaluations. Therefore, this stop point will be used in the next experiments as well.

3.7 CONCLUSIONS

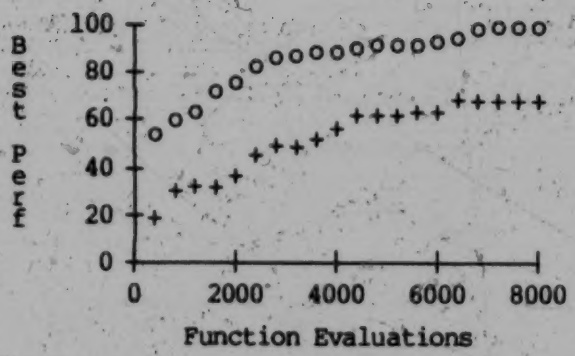
Test Function Set I consists of four one-dimensional, multimodal functions and two 30-dimensional functions. In Hamming space, two of the functions are linearly independent (1 and 4), one is has seven closely linked dependent loci (5), two have distantly linked dependent loci (2 and 6), and one, the damped sinusoid (7), is difficult to analyze, but undoubtedly nonlinear.

The default genetic algorithm exhibited better overall best performance on Functions 1, 2, and 4 than random search, but was worse on Functions 5, 6, and 7. Overall best performance was improved at the expense of online performance by increasing population size to 200.

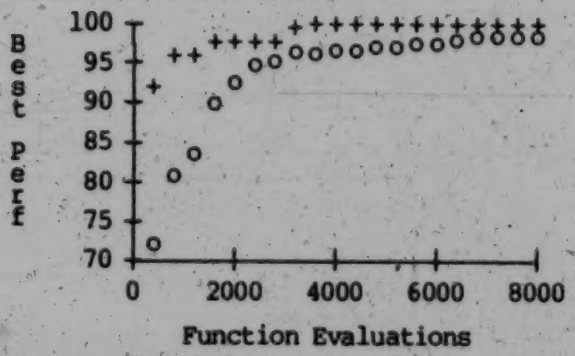
Table 3.6. Experiment 2: Population Size and Mutation Rate.
Online Performance after 8000 Function Evaluations
(results averaged over 4 runs)

Function	Population Size	Mutation Rate .001	Mutation Rate .01	Mutation Rate .02
1	50	79.60	66.51	52.90
	100	69.62	58.83	49.30
	200	57.05	50.13	43.67
2	50	40.39	23.80	10.18
	100	28.79	18.10	8.76
	200	10.24	2.43	-3.62
4	50	83.77	65.14	53.82
	100	73.38	60.28	52.05
	200	61.23	52.42	46.10
5	50	75.26	66.82	54.07
	100	82.82	68.41	62.33
	100	63.55	48.38	46.01
6	50	58.34	30.90	13.45
	100	43.07	18.61	11.78
	200	23.35	15.96	11.05
7	50	80.07	70.85	51.95
	100	70.20	60.60	51.73
	200	59.70	48.93	39.04

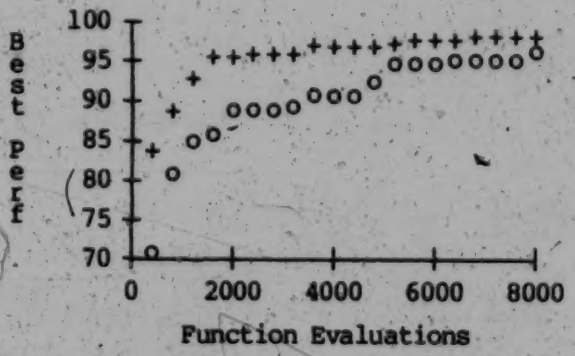
Source of Variation	M.S.	D.F.	F-ratio	Significant
Function	15900.00	5	570.00	*
Population Size	5359.00	2	192.00	*
Mutation Rate	9218.00	2	507.00	*
Function x Pop. Size	117.50	10	4.21	*
Function x Mut. Rate	579.30	10	3.19	*
Pop. Size x Mut. Rate	336.60	4	18.50	*
Function x Pop. Size x Mut. Rate	37.73	20	2.08	*
Within Cell	27.88	54		
Mut. Rate x Within Cell	18.17	108		



a. (o) Function 1
(+) Function 2



b. (o) Function 4
(+) Function 5



c. (o) Function 6
(+) Function 7

Figure 3.10. Experiment 2: Population Size and Mutation Rate. Overall Best Performance for n = 200, P_m = 0.001

CHAPTER 4

PARENT SELECTION

Parent selection in a genetic algorithm is usually accomplished by imposing a probability distribution on the population which reflects the fitness of each individual. The set of parents is then obtained by sampling from the distribution. Section 4.1 discusses the formation of the distribution. Section 4.2 studies the problem of accurate sampling.

4.1 CONSTRUCTING THE PROBABILITY DISTRIBUTION

A discrete probability distribution over n points is defined as a set of values p_i, i=1,...,n, such that

(1) $\sum_{i=1}^{n} p_i = 1$, and

(2) $0 \leq p_i \leq 1$ for i=1,...,n.

The obvious candidate for the probability of selection of an individual s_i in the population is the individual's relative fitness, or $p_i = \frac{f(s_i)}{\sum_{s \in M} f(s)}$. Relative fitnesses satisfy (1) above, but will satisfy (2) only if all fitnesses are non-negative. A fitness value may be any real number, so relative fitnesses might not constitute a probability distribution.

In order to create a distribution which reflects relative fitness, it is necessary to define a base, and measure fitness with respect to this base, yielding

$$p_i = \frac{f(s_i)-\text{base}}{\sum_{s \in M} [f(s)-\text{base}]}.$$

If $f(s_i) \geq$ base for $i=1,\ldots,n$, these p_i will form a valid probability distribution. Obviously, if base = 0 the p_i are simply relative fitnesses.

Since the range of fitnesses encountered is dependent on the function being optimized, a fixed base is impossible without *a priori* knowledge. Instead, the genetic algorithm should define a base at each generation such that $f(s_i) \geq$ base for the current population. One choice for a base is the worst fitness value in the current population. This would be the largest base possible. Another natural choice is the worst value over the last k evaluations, for any desired $k > n$. Finally, the worst value over all evaluations could be used.

Let T be the total number of individuals evaluated up to the time of creation of a distribution. Let T_{max} be the maximum number of function evaluations in the execution of the algorithm (T_{max} is not necessarily known). Then the choices above reduce to

$$\text{base} = \min_{i=j,T} \{f(s) \text{ at time } i\}$$

where $j = \max\{1, T+1-k\}$ and $n \leq k \leq T_{max}$. The choice of k determines which base is employed, that is, how much of the

algorithm history is considered in adjusting the fitnesses.

The space needed for storage of the worst values for $k < T_{max}$ is proportional to $\frac{k}{g}$. Evaluation of the base occurs after each generation of g new individuals, so the worst values for $\frac{k}{g}$ successive sets of g evaluations must be saved. If $k = T_{max}$, only the worst value overall need be maintained.

In Section 4.3, genetic algorithms using the extreme values of $k = n$ (the base is the current population worst) and $k = T_{max}$ (the base is the all-time worst) will be compared. These two values for the base require minimum storage and computation time. The comparison should reveal the effects on performance of the choice of base for the probability distribution.

4.2 SAMPLING FROM THE DISTRIBUTION

Once the probability distribution has been constructed, the set of parents is formed by sampling from the distribution. It is important that the sample taken reflect the distribution as closely as possible. If individuals of low utility are sampled too often, selection according to fitness is impaired. If individuals of high utility are oversampled, premature convergence results.

Clearly, a sampling method must be employed which is accurate. Many sampling methods are available in the literature (Kleijnen, 1974; Burt, Gaver, and Perlas, 1970; Hammersley and Handscomb, 1964; Kahn and Marshall, 1953). Most of these define accuracy of sampling as reduction of

variance. Before comparing the effects of several sampling methods on genetic algorithm behavior, it is necessary to define accuracy of sampling for the parent selection process.

4.2.1 Measures of Accuracy

As in Section 4.1, let p_i be the probability of selection of individual s_i in the population, i=1,...,n. For ease of notation, let m = n' be the size of the sample to be taken. Let e_i be the expected number of times s_i will occur in the sample. Ideally, s_i should be sampled mp_i times, or

$$e_i = mp_i, \quad i=1,\ldots,n.$$

Let f_i be the actual frequency, or number of times, s_i is sampled by a given method SM. Then f_i is an estimator for e_i.

The frequency of sampling of each s_i is of major importance in a genetic algorithm. For this reason, sampling methods should be analyzed for the accuracy of their sampling frequency estimators, rather than for the usual accuracy of the mean estimator.

The accuracy of an estimator is measured by bias and variance. For each f_i, i=1,...,n,

$$\text{bias}[f_i] = |E[f_i] - e_i|$$

$$\text{var}[f_i] = E[f_i - E[f_i]]^2.$$

An accurate estimator minimizes both bias and variance.

It will be hard to evaluate sampling methods if n estimators are measured separately for bias and variance. The biases of the f_i can be summed to give the bias of the method:

$$\text{bias}_{SM} = \sum_{i=1}^{n} \text{bias}[f_i],$$

and likewise for the variance:

$$\text{var}_{SM} = \sum_{i=1}^{n} \text{var}[f_i].$$

For a general measure of estimator accuracy, however, bias and variance can be combined into mean square error:

$$\begin{aligned} MSE[f_i] &= \text{var}[f_i] + \text{bias}^2[f_i] \\ &= E[f_i - e_i]^2 \end{aligned}$$

Let the mean square error of frequencies for a sampling method SM be

$$MSE_{SM} = \sum_{i=1}^{n} MSE[f_i].$$

This mean square error will be used to compare sampling methods for genetic algorithms, but bias and variance will be considered separately as well.

4.2.2 Stochastic Sampling with Replacement

The simplest way to produce a random sample from a probability distribution is by stochastic, or Monte Carlo, sampling with replacement. First, the cumulative distribution is computed. Each sample element is then obtained by generating a uniform random number between 0 and 1 and sampling the point corresponding to this number on the

cumulative distribution.

Let random() be a function generating uniform(0,1) numbers. Assume that the sample is to be placed in random order in $sample_j$, j=1,...,m. Stochastic sampling with replacement (method S) is displayed in pidgin Algol in Figure 4.1.

```
begin method S:
   cumdist(1) = p(1)
   for i from 2 to n do
      cumdist(i) = cumdist(i-1) + p(i)
      enddo
   for j from 1 to m do
      r = random()
      for i from 1 to n do
        if r<cumdist(i) then breakloop endif
        enddo
      sample(j) = i
      enddo
end method S.
```

Figure 4.1. Stochastic Sampling Method

Under stochastic sampling with replacement, f_i has a binomial distribution, or

$$P(f_i=y) = \binom{m}{y} p_i^{\,y} (1-p_i)^{m-y}, \quad \text{for } y = 0,\ldots,m.$$

Then

$$E[f_i] = mp_i$$

$$bias[f_i] = 0$$

$$var[f_i] = mp_i(1-p_i)$$

$$MSE_S = \sum_{i=1}^{n} mp_i(1-p_i) = \sum_{i=1}^{n} e_i\left(1-\frac{e_i}{m}\right)$$

Stochastic sampling is unbiased. Its mean square error is composed entirely of variance.

4.2.3 Deterministic Sampling

The following deterministic technique has been referred to as the "fixed sequencing method" by Ehrenfeld and Ben-Tuvia (1962) and also as "selective sampling" by Brenner (1963).

For each point i, the expected sampling frequency $e_i = mp_i$ is computed. Each point is allotted samples according to the integer portion of this frequency for a total of $W \leq m$ sample elements. The n points are then ordered so that the fractional parts of their expected frequencies are monotonically decreasing, and the first $R = m-W$ points are allotted one more sample element each, to yield a total sample size of m. A selection sort is used to find the first R points in order, on the assumption that R will be reasonably small and only a partial sort will be needed.

Once the number of allotments for each point in the distribution is known, a randomly ordered sample must be obtained. This can be done in two ways. 1) Sample stochastically, from the distribution of allotments, readjusting the cumulative distribution after each selection (Brenner, 1963). 2) Create a list of sample elements. Include point i x times if it has been allotted x samples. Choose the jth element in the final sample by generating a uniform random number between 1 and m+1-j, using that list entry and moving the m+1-jth entry to replace the one used. The second technique requires m units of space for the

temporary array of elements. The first requires n units of space for the number of samples allotted to each point in the distribution, plus a considerable amount of time to compute the cumulative distribution repeatedly. For this reason, the second ordering technique will be used. The deterministic method (method D) is shown in Figure 4.2.

```
begin method D:
   W = 0
   for i from 1 to n do
      expected = m * p(i)
      intexpected = truncate(expected)
      fraction(i) = expected - intexpected
      for dummy from 1 to intexpected do
         W = W + 1
         list(W) = i
         enddo
      enddo
   for j from W+1 to m do
      max = 1
      for i from 2 to n do
          if fraction(i) > fraction(max) then max = i endif
          enddo
      list(j) = max
      fraction(max) = 0
      enddo
   comment order the sample
   remaining = m
   for j from 1 to m do
      r = truncate(remaining * random()) + 1
      sample(j) = list(r)
      list(r) = list(remaining)
      remaining = remaining - 1
      enddo
end method D.
```

Figure 4.2. Deterministic Sampling Method

Let w_i be the integral part of e_i, and r_i the fractional part, i.e.

$$w_i = \lfloor e_i \rfloor$$

$$r_i = e_i - w_i.$$

Define

$$W = \sum_{i=1}^{n} w_i$$

$$R = \sum_{i=1}^{n} r_i,$$

Note that W + R = m. Assume that the n points are labelled so that $r_i \geq r_{i+1}$, i=1,...,n-1.

For i=1,...,R,

$$f_i = w_i + 1$$

$$E[f_i] = w_i + 1 = f_i$$

$$\text{bias}[f_i] = w_i + 1 - e_i > 0$$

$$\text{var}[f_i] = E[f_i - E[f_i]]^2 = 0$$

For i=R+1,...,n,

$$f_i = w_i$$

$$E[f_i] = w_i = f_i$$

$$\text{bias}[f_i] = w_i - e_i \quad (= 0 \text{ only if } e_i \text{ is an integer})$$

$$\text{var}[f_i] = 0$$

The deterministic method is unbiased only in the special case when all e_i are integers. The mean square error will be

$$MSE_D = \sum_{i=1}^{n} (f_i - e_i)^2$$

$$= \sum_{i=1}^{R} (w_i + 1 - e_i)^2 + \sum_{i=R+1}^{n} (w_i - e_i)^2$$

$$= \sum_{i=1}^{R} (1 - r_i)^2 + \sum_{i=R+1}^{n} r_i^2$$

MSE_D reflects the bias of this method, as there is no variance in a non-stochastic technique.

4.2.4 Remainder Stochastic Sampling with Replacement

The third sampling technique is a combination of the first two. Stochastic sampling has no bias but high variance, whereas the deterministic approach is biased but has no variance. Remainder stochastic sampling combines the desirable characteristics of both methods.

In the remainder stochastic method, the expected frequencies of sampling are computed as in the deterministic method, and each point is allotted sample elements according to the integer part of this frequency. But then, instead of ordering the fractions, a new probability function is formed over the n points using the $\frac{r_i}{R}$ as the new p_i. The R remaining elements are sampled by the stochastic method according to this new function. Remainder stochastic sampling with replacement (method RS) is illustrated in Figure 4.3.

When all e_i are integers, this method is equivalent to the deterministic method. Since this is not usually the case, consider the method when $R > 0$.

Examine the stochastic stage of sampling which is done over the function based on the r_i's. Let

$$f_i = w_i + g_i,$$

where g_i is the number of times i is selected during the stochastic sampling. g_i is binomially distributed. Thus

$$P(g_i=y) = \binom{R}{y} \left(\frac{r_i}{R}\right)^y \left(1 - \frac{r_i}{R}\right)^{R-y} \quad \text{for } y=0,\ldots,R.$$

```
begin method RS
   W = 0
   for i from 1 until n do
      expected = m * p(i)
      intexpected = truncate(expected)
      fraction(i) = expected - intexpected
      for dummy from 1 to intexpected do
           W = W + 1
           list(W) = i
           enddo
      enddo
   R = m - W
   if R = 0 then stop endif
   for i from 2 to n do
      fraction(i) = fraction(i-1) + fraction(i)
      enddo
   for j from W+1 to m do
      r = R * random()
      for i from 1 to n do
           if r<fraction(i) then breakloop endif
           enddo
      list(j) = i
      enddo
   remaining = m
   for j from 1 until m do
      r = truncate(remaining * random()) + 1
      sample(j) = list(r)
      list(r) = list(remaining)
      remaining = remaining - 1
      enddo
end method RS.
```

Figure 4.3. Remainder Stochastic Sampling Method

It follows that

$$E[g_i] = R(\frac{r_i}{R}) = r_i$$

$$var[g_i] = R(\frac{r_i}{R})(1 - \frac{r_i}{R}) = r_i(1 - \frac{r_i}{R})$$

Using these values for g,

$$E[f_i] = E[w_i + g_i] = w_i + r_i = e_i$$

$$bias[f_i] = 0$$

$$var[f_i] = E[f_i - E[f_i]]^2$$

$$= E[w_i + g_i - E[w_i + g_i]]^2$$

$$= E[g_i - E[g_i]]^2$$

$$= \mathrm{var}[g_i]$$

$$= r_i(1 - \frac{r_i}{R})$$

$$MSE_{RS} = \sum_{i=1}^{n} r_i(1 - \frac{r_i}{R})$$

4.2.5 Stochastic Sampling Without Replacement

Stochastic sampling with replacement has no bias, but does have variance. It may be possible to reduce the variance of stochastic sampling by sampling without replacement from the expected frequencies e_i.

In sampling without replacement, a random element is sampled from a set and that element is then removed from the set before the next sample is taken. Given a probability distribution p_i over n points, sampling without replacement is ordinarily accomplished by computing $e_i = mp_i$, $i=1,\ldots,n$, sampling from these e_i, and reducing e_i by one every time element i is sampled. Unfortunately, e_i can be reduced to at most zero after a sample, so if $e_i < 1$, the amount of reduction can be at most e_i. Sampling once from an element i which has $e_i < 1$ can introduce bias into the method.

Consider the simple case where $n = m = 2$ and $p_1 > p_2 > 0$. Initially $e_1 = 2p_1 > 1$ and $e_2 = 2p_2 < 1$. The expected value of f_i is the sum over all samples of the probability that i is taken as that sample.

$$E[f_1] = p_1 + P(1 \text{ is second sample})$$
$$= p_1 + p_1 P(1 \text{ is second, given 1 is first})$$
$$+ p_2 P(1 \text{ is second, given 2 is first})$$
$$= p_1 + p_1(2p_1 - 1) + (1-p_1)(1)$$
$$= 2p_1^2 - p_1 + 1$$
$$< 2p_1, \text{ since } 0.5 < p_1 < 1.$$

$$E[f_2] = p_2 + p_1(2p) + p_2(0)$$
$$= p_2 + (1-p_2)(2p_2)$$
$$= p_2(3-2p_2).$$
$$> 2p_2, \text{ since } p_2 < 0.5.$$

The method in this case is biased towards the overselection of the element with the smaller probability.

Sampling without replacement can also be applied in the remainder stochastic method. In this case $e_1 = r_i$ initially for i=1,...,n. As shown in Appendix B, this method will be biased unless $r_i = r_j$ for all r_i and $r_j \neq 0$. The method of sampling without replacement where initially $e_i = mp_i$ will almost always be biased. This can be proved for remainder stochastic sampling without replacement. Stochastic sampling without replacement will probably be biased unless the e_i are integers for all i=1,...,n.

4.3 COMPARISONS

The mean square errors for the deterministic (D), remainder stochastic with replacement (RS), and stochastic with replacement (S) methods are ordered

$$MSE_D \leq MSE_{RS} \leq MSE_S$$

(see Appendix C for proofs). Furthermore, the strict

inequalities

$$MSE_D < MSE_{RS} < MSE_S$$

hold unless

1. $e_i < 1$ for all i,
2. R = 0, or
3. R = 1 and $r_i = r_j$ for all $r_i, r_j \neq 0$.

Method RS has the least variance of the two unbiased methods S and RS. Although the two methods of stochastic and remainder stochastic sampling without replacement have not been analyzed for mean square error, they are biased.

Brenner (1963) has shown that the deterministic method has the minimum mean square error of any possible sampling method. Therefore if bias is acceptable, deterministic sampling should be preferable to any of the stochastic methods.

4.3.1 Experiment 3: Sampling Methods

The mean square errors of stochastic, remainder stochastic and deterministic sampling are usually unequal. Experiment 3 was designed to show the effects of differences in accuracy among the three methods. Samples of sizes m = 50, 100, and 200 were taken from a population of n = 50 points according to uniform random distributions with variances of 0.01, 1, and 100. The factors were

1. Variance of the Distribution (0.01, 1, 100)
2. Replications (10)
3. Sample Size (50, 100, 200)
4. Sampling Method (S, RS, D).

Results are shown in Table 4.1. The mean square error of the deterministic method is far lower than that of the other methods. As expected, the mean square error of method RS is limited by n, while that of method S grows as m grows. Resource requirements and relative accuracies for the three methods are summarized in Table 4.2. If m = 2n, as is the case for parent selection sampling in a genetic algorithm, space and time requirements for the algorithms are all of the same order of complexity.

4.3.2 Experiment 4: Parent Selection Methods

It remains to be seen whether the large differences in accuracy among these sampling methods will affect the performance of a genetic algorithm. In Experiment 4, six sampling methods were run on Test Function Set I using two values of the base for the probability distribution. The algorithms employed a population size of 200, $P_m = 0.001$, and $P_c = 0.6$. The factors were

1. Function (1, 2, 4, 5, 6, 7)
2. Sampling Method (deterministic, remainder stochastic without replacement, stochastic without replacement, remainder stochastic with replacement, stochastic with replacement, ranking method)
3. Replications (4)
4. Base of the Probability Distribution (worst all-time, current population worst).

The last method of selection is a ranking method proposed by

Table 4.1. Experiment 3: Sampling Methods.
Mean Square Error
(results averaged over 10 runs on populations of size 50)

Population Variance	Sample Size (m)	Method		
		Deter-ministic	Remainder Stochastic	Stochastic
.01	50	7.8	45.9	50.2
	100	8.1	45.0	101.8
	200	8.4	51.8	200.7
1	50	7.9	47.4	54.6
	100	8.1	48.0	103.2
	200	8.3	46.8	199.6
100	50	8.3	45.7	52.4
	100	8.4	48.9	91.8
	200	9.2	45.2	211.0

Source of Variation	M.S.	D.F.	F-ratio	Significant
Population Variance	5.6	2	.03	
Sample Size	61830.0	2	307.00	*
Sampling Method	280500.0	2	1410.00	*
Var. x Sample Size	104.6	4	.52	
Var. x Sampling Method	12.0	4	.06	
Size x Method	59283.0	4	281.00	*
Var. x Size x Method	190.8	8	.90	
Within Cell	194.2	27		
Size x Within Cell	201.1	54		
Method x Within Cell	199.1	54		
Size x Method x x Within Cell	211.0	108		

Table 4.2. Comparison of Sampling Methods.

Method	Mean Square Error	Biased	Space[a]	Time[b] Worst	Time[b] Average
Stochastic	high	no	n	O(mn)	O(mn)
Remainder Stochastic	moderate	no	n+m	O(mn)	$O(n^2)$
Deterministic	low	yes	n+m	O(mn)	$O(n^2)$

a: n = number of points in the distribution,
m = sample size.

b: worst case time assumes R = m-1,
average case time assumes R = n/2.

Art Wetzel (1979) which differs radically from the other selection methods. Each parent is selected by sampling twice from the distribution (stochastic sampling with replacement) and using the better of the two individuals.

Table 4.3 shows the results of Experiment 4. The sampling method is significant, but not to a great degree. Remainder stochastic sampling (with or without replacement) is usually inferior. Deterministic sampling and stochastic sampling without replacement are often better than the others. It is interesting to note that both of these methods are biased. Apparently low mean square error is preferable to lack of bias in the sampling process.

Preliminary experiments indicated that the choice of base would also be significant. Those experiments were run on populations of 50 individuals for 200 generations. Using the current population worst as the base hastened convergence considerably, often to a sub-optimal value. Apparently over a larger population, with a run length of only 40 generations, the choice of base is less important, because the difference between all-time worst and current population worst values is small.

The effects of sampling method and distribution base on population variance are shown in Table 4.4. The all-time worst value base maintained population variance significantly more than the base of the current population worst. Sampling by the deterministic method and stochastic sampling without replacement also maintained population

Table 4.3. Experiment 4: Parent Selection Methods.
Overall Best Performance after 8000 Function Evaluations
(results averaged over 4 runs)

Function	Base[a]	Sampling Method[b]					
		D	RSW	SW	RS	S	RA
1	A	97.00	100.00	100.00	94.00	100.00	100.00
	C	100.00	100.00	100.00	100.00	100.00	100.00
2	A	67.50	61.25	67.50	58.75	70.00	63.75
	C	67.50	58.75	65.00	66.25	63.75	65.00
4	A	98.79	98.77	99.32	95.74	98.76	99.76
	C	99.74	99.05	99.75	98.91	99.47	99.84
5	A	100.00	98.61	100.00	99.67	97.62	97.36
	C	100.00	97.03	99.94	97.35	99.65	100.00
6	A	92.12	94.51	94.59	89.45	94.09	90.95
	C	96.35	92.02	95.29	88.84	97.03	90.61
7	A	100.00	96.26	98.34	94.49	96.13	98.35
	C	98.19	93.48	99.04	93.38	93.63	96.18

a: base of the probability distribution
A = all-time worst value
C = current population worst

b: sampling methods
D = deterministic
RSW = remainder stochastic without replacement
SW = stochastic without replacement
RS = remainder stochastic with replacement
S = remainder stochastic with replacement
RA = ranking method

Source of Variation	M.S.	D.F.	F-ratio	Significant
Function	8851.00	5	450.00	*
Sampling Method	90.76	5	4.61	*
Distribution Base	3.21	1	.26	
Function x Method	18.04	25	.92	
Function x Base	14.86	5	1.21	
Sampling Method x Base	19.01	5	1.55	
Function x Method x Base	12.50	25	1.02	
Within Cell	19.68	108		
Base x Within Cell	12.30	108		

Table 4.4. Experiment 4: Parent Selection Methods.
Average Maximum Frequency after 8000 Function Evaluations
(results averaged over 4 runs)

Function	Base[a]	Sampling Method[b]					
		D	RSW	SW	RS	S	RA
1	A	.838	.964	.904	.982	.882	.999
	C	.999	.999	.999	.999	.999	.999
2	A	.879	.946	.913	.976	.892	.980
	C	.970	.979	.967	.986	.982	.988
4	A	.728	.957	.757	.980	.784	.999
	C	.835	.945	.829	.969	.879	.999
5	A	.718	.834	.725	.934	.763	.952
	C	.760	.848	.727	.953	.802	.977
6	A	.786	.950	.681	.946	.788	.927
	C	.796	.978	.865	.995	.850	.999
7	A	.736	.867	.763	.958	.752	.978
	C	.777	.866	.744	.934	.777	.955

a: base of the probability distribution
A = all-time worst value
C = current population worst

b: sampling methods
D = deterministic
RSW = remainder stochastic without replacement
SW = stochastic without replacemet
RS = remainder stochastic with replacement
S = remainder stochastic with replacement
RA = ranking method

Source of Variation	M.S.	D.F.	F-ratio	Significant
Function	.14580	5	88.80	*
Sampling Method	.25820	5	157.00	*
Base	.12500	1	97.10	*
Function x Method	.01264	25	7.69	*
Function x Base	.08721	5	6.78	*
Sampling Method x Base	.01200	5	9.33	*
Function x Method x Base	.00290	25	2.26	*
Within Cell	.00164	108		
Base x Within Cell	.00129	108		

variance, which no doubt contributed to their superior overall best performance. The deterministic method, using the all-time worst as base, provided consistently lower average maximum frequency than all other selection methods tested.

To complete the analysis, the overall best performance of the two best sampling methods was plotted over time (see Figures 4.4a and 4.4b). Each point was averaged over four runs using the all-time worst value as base. With the exceptions of Functions 5 and 7, the early performances of the two algorithms were identical. On Functions 5 and 7, the differences were small and short-lived. This means that the greater average maximum frequency of stochastic sampling without replacement was not a symptom of faster convergence to a good value. Since overall best performance of the two methods is similar and deterministic sampling provides more population variance, the deterministic method of sampling will be employed in the remaining experiments.

4.4 CONCLUSIONS

The parent selection probability distribution can be constructed according to

$$p_i = \frac{f(s_i)-\text{base}}{\sum_{s \in M} [f(s)-\text{base}]}.$$

The use of all-time worst value as base is best at maintaining population variance.

The sampling of parents according to this distribution must be accurate. Remainder stochastic sampling with replacement has the lowest variance for an unbiased method. Deterministic sampling, although biased, provides the greatest accuracy in terms of mean square error of sampling frequencies. Deterministic sampling yields greater population variance than the other methods when employed in a genetic algorithm. The deterministic and stochastic without replacement methods are good with respect to overall best performance, although choice of sampling method alone did not affect performance greatly.

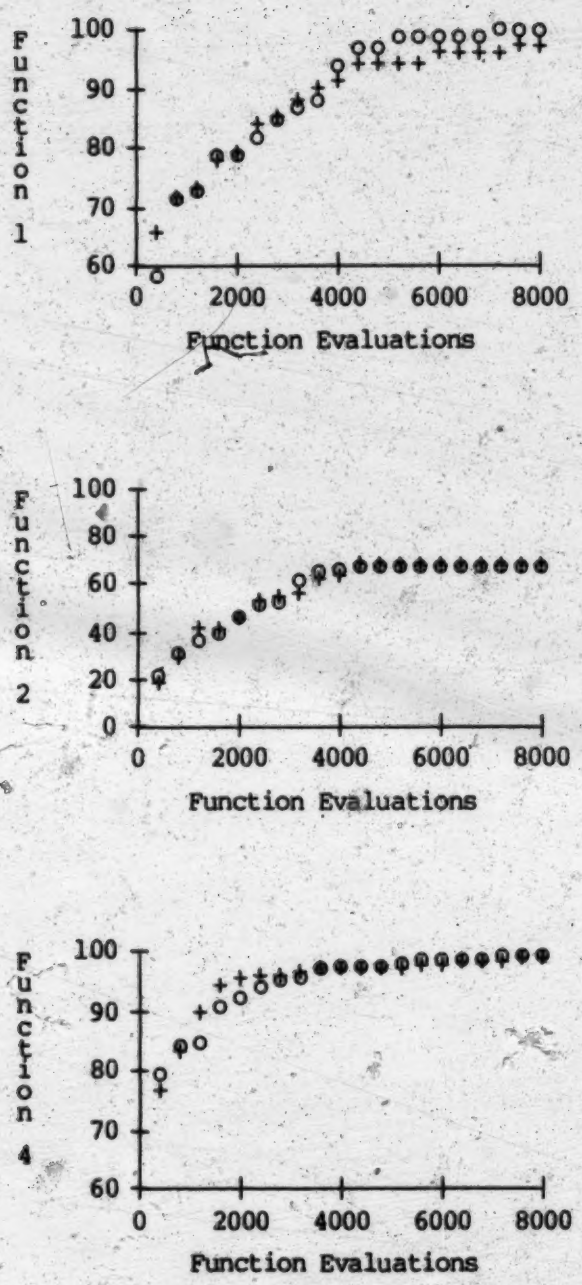
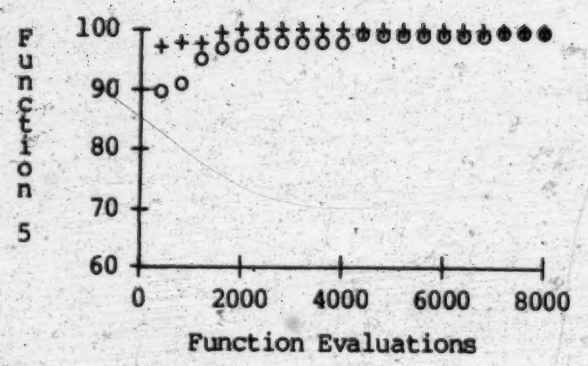
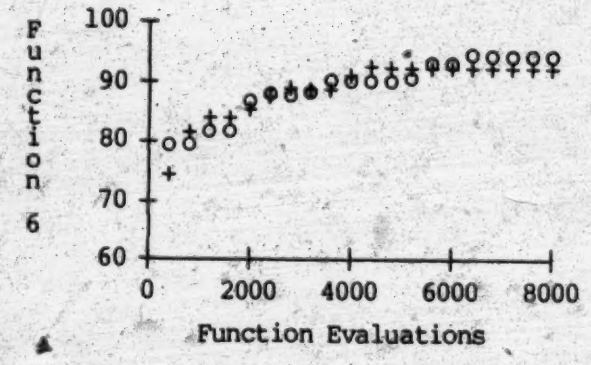


Figure 4.4a. Experiment 4: Parent Selection Methods.
Overall Best Performance on Functions 1, 2, and 4
(o) Stochastic Sampling without Replacement
(+) Deterministic Sampling





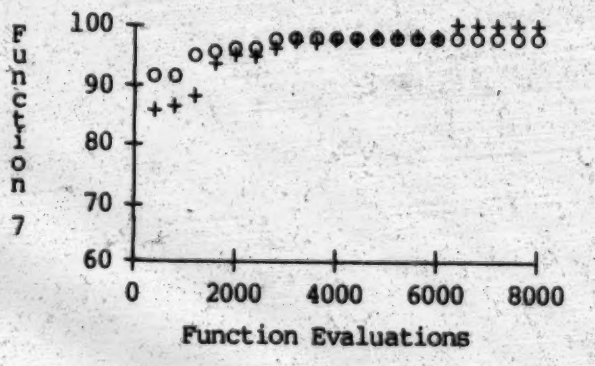


Figure 4.4b. Experiment 4: Parent Selection Methods.
Overall Best Performance on Functions 5, 6, and 7
(o) Stochastic Sampling without Replacement
(+) Deterministic Sampling

CHAPTER 5

DIPLOIDY AND DOMINANCE

This chapter investigates diploidy and several dominance schemes as mechanisms for reducing loss of population variance. First, the natural systems involving diploidy will be examined, and then a set of dominance schemes for genetic algorithms will be proposed.

5.1 NATURAL DIPLOIDY

Diploid organisms have chromosomes in homologous pairs. There may be many such pairs in a higher organism. The following discussion assumes organisms with only one homologous pair; however, the concepts are easily generalized to multiple pairs.

A diploid organism is <u>homozygous</u> for an allele a_0 at a locus if both chromosomes contain a_0 at that locus. If the two chromosomes contain different alleles at the locus, the organism is <u>heterozygous</u>. Often in a heterozygous individual, the phenotype will be determined using only one of the alleles at a locus. This allele is said to be <u>dominant</u> over the allele not used. That allele is called <u>recessive</u>.

5.1.1 The Reproductive Cycle

For haploid organisms, the cycle of reproduction is fairly simple. An organism produces an exact copy of itself

via mitotic division. This copy may then recombine with the copy of another parent (see Figure 5.1a). Diploid organisms undergo meiotic division (Figure 5.1b) followed by zygote formation. In meiotic division, both of the homologous chromosomes are duplicated. Crossover occurs among the four chromosomes in the parent. A zygote (genetically complete diploid organism) is then formed by pairing one of the chromosomes with a chromosome from another parent. The offspring always contains genetic information from both parents.

5.1.2 Recombination

After duplication there are four chromosomes in a parent. Of these, two rarely undergo crossover, while the other two recombine with a frequency proportional to chromosome length. If the probability of crossover between the middle two chromosomes is P_c', then the probability that the one chromosome in the offspring is the result of crossover is $0.5 \cdot P_c'$. The meiotic cycle can be simulated by using a crossover probability of $P_c = 0.5 \cdot P_c'$, for $0 \leq P_c \leq 0.5$, and omitting the duplication step (Figure 5.1c).

In haploid populations, crossover is the main mechanism for recombining alleles in the genotype. In diploid populations, however, zygote formation recombines chromosomes to create new genotypes. Consider a haploid population with crossover rate P_c and no mutation. The probability that an offspring is identical to one of its

parents is equal to the probability of no crossover plus the probability that crossover occurs but the parents are identical on one side of the crossover point.

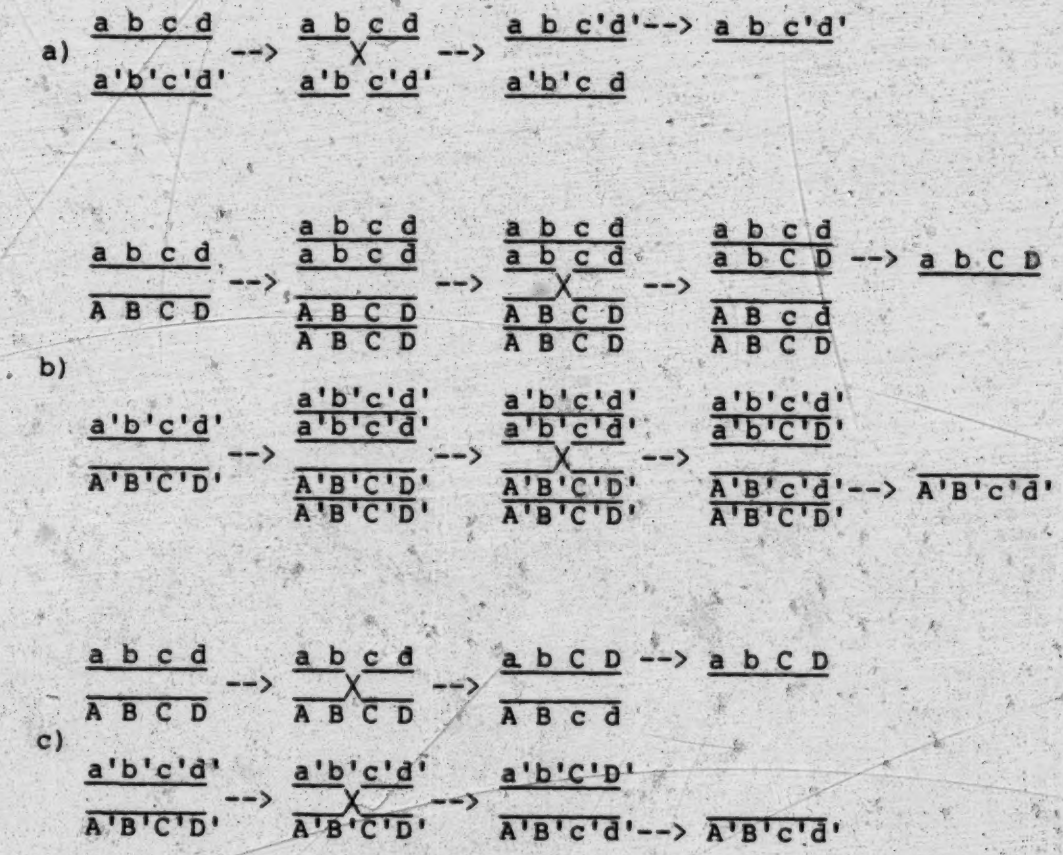


Figure 5.1. a) Mitotic Division (Haploid Organisms)
b) Meiotic Division (Diploid Organisms)
c) Simulation of Meiotic Division

In other words, the probability of no visible recombination is

$$P_{haploid}(N.V.R.) = (1 - P_c) + P_c P(\text{identical segments}).$$

Now consider a diploid population containing copies of unique chromosomes C_i, i=1,...,k. Assume that there is no crossover or mutation. Define the probability of no visible recombination as the probability that an offspring is an exact copy of one of its parents.

Suppose a parent having chromosomes c_1 and c_2 mates with a parent having chromosomes c_3 and c_4. If the offspring receives chromosomes c_1 and c_3, for example, it will be a copy of one parent if $c_1 = c_4$ or if $c_2 = c_3$. In general,

$$P_{diploid}(N.V.R.) = 2P(c_i = c_j),$$

for any two chromosomes c_i, c_j sampled from the population. If the chromosomes are distributed randomly throughout the population and the individuals mate randomly, this becomes

$$P_{diploid}(N.V.R.) = 2\sum_{i=1}^{k} p(C_i)^2.$$

Assume that $p(C_i) = \frac{1}{k}$ for i=1,...,k, that is, that the k chromosome types occur in equal numbers. Then

$$P_{diploid}(N.V.R.) = \frac{2}{k}.$$

If the number of chromosome types in the population is greater than five, or $k > 5$,

$$P_{diploid}(N.V.R.) = \frac{2}{k} < 0.4 < P_{haploid}(N.V.R.),$$

when P_c for a haploid population is 0.6.

If $p(C_i) \neq \frac{1}{k}$ for i=1,...,k, then the probability of no visible recombination will increase. But unless the population has very little variety in its chromosomes, fertilization alone will provide as much recombining as

crossover provides in a haploid population.

It remains desirable to recombine via crossover in diploid populations. However, high crossover rates ($P_c > 0.5$) for the simulated reproductive cycle are not needed for effective recombination, and are not found in natural diploid organisms. Drosophila Melanogaster (a fruit fly) and Neurospora Crassa (a fungus) exhibit crossover in 5% to 50% of their offspring chromosomes (Strickberger, 1976). Only very short chromosomes show less than 30% crossover. It would seem reasonable to place P_c' between 0.6 and 1.0, yielding $0.3 \leq P_c' \leq 0.5$.

5.1.3 Mutation

The crossover rate needed for recombination in diploid organisms is lower than that needed in haploid organisms. The mutation rate needed to prevent the loss of alleles from the population is also smaller under diploidy with dominance. The following is a modification of the presentation in Holland (1975).

Let the alleles possible at one locus be $a_0, a_1, \ldots, a_B$. Assume that a_0 is the least fit of these on the average. Let $ph(a_0,t)$ be the proportion of the population at time t which displays a_0 in the phenotype. Let the rate of reproduction of a_0 be 1-e for some $0 < e \leq 1$. This means that at time t+1 in a population of size n, it is expected that $2n(1-e)ph(a_0,t)$ parents will be selected which have a_0 employed in the phenotype.

Let $p(a_0,t)$ be the proportion of the chromosomes in the population at time t which contain a_0. To guarantee that a_0 never disappears from the population, a mutation rate is needed such that if $p(a_0,t) = \frac{1}{N}$, then $p(a_0,t+1) \geq \frac{1}{N}$, where N is the total number of chromosomes in the population. The expected number of a_0's in the population at time t+1 is equal to the number which survive evaluation in the phenotype, plus the number which are masked in other phenotypes, plus the number which mutate to a_0, minus the number which mutate away from a_0. The expected number of a_0's in the population is computed as

$$N \cdot P(a_0,t+1) = n \cdot (1-e) \cdot ph(a_0 t) \cdot (\text{number of } a_0\text{'s in each organism with a phenotype using } a_0)$$
$$+ n \cdot (\text{average reproductive rate for alleles not } a_0) \cdot (\text{proportion of genotypes containing masked } a_0\text{'s}) \cdot (\text{number of } a_0\text{'s masked})$$
$$+ N \cdot P_m [1 - p(a_0,t)]$$
$$- N \cdot P_m \cdot p(a_0,t).$$

In a haploid population, $N = n$, $p(a_0,t) = ph(a_0,t)$, and no masked alleles are carried in a genotype. Therefore

$$n \cdot p(a_0,t+1) = n(1-e)p(a_0,t) + nP_m(1 - p(a_0,t)) - nP_m p(a_0,t)$$

$$p(a_0,t+1) = (1-e)p(a_0,t) + P_m(1 - 2p(a_0,t)).$$

If $p(a_0,t) = \frac{1}{N}$, then $p(a_0,t+1) \geq \frac{1}{N}$ if

$$P_m \geq \frac{e}{n-2}.$$

If a_0 has a very low average fitness, it will have a

reproductive rate of 0, or e will equal 1. The mutation rate needed to insure the presence of very bad alleles under haploidy is

$$P_m \geq \frac{1}{n-2} \approx \frac{1}{n}.$$

In a diploid population, N = 2n. Assume that the alleles are uniformly distributed over the population. Let $a_1, a_2, \ldots, a_B$ each be dominant over a_0. Then a_0 will be used in the phenotype whenever the individual is homozygous for a_0.

$$ph(a_0,t) = p(a_0,t)^2.$$

The proportion of phenotypes heterozygous for a_0 will be

$$2p(a_0,t)[1 - p(a_0,t)].$$

The average reproductive rate for alleles other than a_0 is

$$1 + e\left(\frac{ph(a_0,t)}{1-ph(a_0,t)}\right).$$

The expected number of a_0's at time t+1 is

$$\begin{aligned} 2 \cdot n \cdot p(a_0,t+1) &= n(1-e)p(a_0,t)^2 \quad (2) \\ &+ n\left(1 + e\frac{p(a_0,t)^2}{1-p(a_0,t)^2}\right)2p(a_0,t)[1 - p(a_0,t)] \\ &+ 2nP_m[1 - 2p(a_0,t)]. \end{aligned}$$

Simplifying,

$$p(a_0,t+1) = p(a_0,t) - e\left(\frac{p(a_0,t)^2}{1+p(a_0,t)}\right) + P_m[1 - 2p(a_0,t)].$$

If $p(a_0,t) = \frac{1}{N}$, then $p(a_0,t+1) \geq \frac{1}{N}$ if

$$P_m \geq \frac{e}{(N+1)(N-2)}.$$

In the worst case e = 1, so the mutation rate needed under diploidy to maintain poor alleles in the population is

$$P_m \geq \frac{1}{(2n+1)(2n-2)} \approx \frac{1}{4n^2}.$$

5.1.4 A Diploid Reproductive Plan

Diploid organisms employ a different reproductive cycle than the cycle of haploid organisms. Mitosis is replaced by meiosis and fertilization. The observed crossover rate for diploid organisms is 0.3 to 0.5, compared to $P_c = 0.6$ in previous haploid genetic algorithms. The mutation rate needed to maintain alleles in the population is on the order of $\frac{1}{n^2}$, whereas it is $\frac{1}{n}$ for haploid populations.

The following diploid Reproductive Plan will be used for experiments in Chapter 5:

1. 100 diploid organisms are generated according to a uniform random distribution over the set of possible genotypes.
2. 200 parents are selected by deterministic sampling over a distribution reflecting the utility of the phenotypes.
3. 100 parental groups are formed by random pairing. Single crossovers occur between the two chromosomes in each parent with a frequency $0.3 \leq P_c \leq 0.5$. Mutation occurs at each locus with a frequency $P_m = \frac{1}{4n^2} = 0.000025$. One chromosome is chosen at random from each parent, forming one diploid offspring per parental group.
4. The old population is completely replaced by the 100 offspring.
5. The cycle repeats from step 2.

5.2 DOMINANCE SCHEMES

A dominance scheme maps two alleles at a locus onto one value to be used in determining the phenotype. The benefits of diploidy in avoiding loss of population variance would be greatest if the poorer alleles were completely recessive. Such deleterious alleles would be masked in heterozygotes, and thus preserved in the population. If an inferior allele was dominant, it would probably disappear quickly from the population.

There is no evidence that in natural diploid organisms better alleles are intrinsically dominant. In fact, it is not known whether or not there is any relationship between average allele fitness and dominance. Although dominance has been studied by population geneticists (e.g. Watterson, 1977; Ford and Sheppard, 1965; Clark, 1964; Crosby, 1963), no firm conclusions have been drawn about its development or utility. However, it is possible to examine natural dominance mechanisms and construct hypotheses about possible relationships to fitness.

Consider a locus in a diploid organization with alleles $\{a_0, a_1\}$. Assume that this locus specifies a simple trait or phenotype. The values of + and - for the trait are produced by the presence and absence, respectively, of a gene product in the organism. Assume that allele a_1 codes for a sufficient amount of this product, while a_0 codes for nothing. Then the phenotypic values for the three possible genotypes will be:

a_1a_1 --> +

a_1a_0 --> +

a_0a_0 --> -

a_1 is dominant over a_0.

In a genetic algorithm, dominance by presence of a product can be simulated by a random, fixed, global dominance map. Initially, a chromosomal map is generated randomly to indicate which of the alleles at each locus is the producer, or dominant allele. This map applies globally to all individuals in the population and remains fixed throughout time.

In natural organisms, there are mechanisms for the dynamic development of the genotype (Section 2.3.4), including deletion mechanisms. An organism in which + was disadvantageous would be improved by the deletion of the locus. There may be a greater proportion of loci in natural populations in which + is advantageous, due to the deletion of loci producing deleterious products. Thus, despite the lack of an explicit mechanism which relates dominance and fitness, there may be a correlation between dominance and fitness in populations which have developed adaptively.

Since genetic algorithms with binary representations have no deletion operator, the natural mechanism of deletion cannot be modelled explicitly. Rather, the relationship between dominance and allele value caused by deletions must be simulated. The correspondence is simple: the better of two alleles becomes dominant.

The value of an allele is proportional to its frequency of occurrence in the population. Therefore, a variable, global dominance map may be used. At each generation, the probability of an allele being dominant is equal to its frequency of occurrence in the previous generation. Every time a heterozygote is created, its phenotype is determined stochastically according to these probabilities. The probabilities apply globally to the individuals in the population, but vary from generation to generation.

A less expensive variant on the stochastic, variable, global map is a deterministic, variable, global map. At each locus, the allele with the greatest frequency in the previous generation is declared to be dominant. A new dominance map is constructed at each generation, but the determination of dominance for each heterozygote is no longer probabilistic.

The model of two alleles defining the presence or absence of a product can be generalized to two alleles, each producing a product. Let a_1 code for product A and a_0 code for B. One of A or B may be null. If there is no control of expression of the products, the phenotypes will be

$$a_1a_1 \text{ --> } 2A$$
$$a_1a_0 \text{ --> } A+B$$
$$a_0a_0 \text{ --> } 2B$$

There is no dominance in this case. If differences in the quantity of a product do not produce differences in the phenotypic trait, the result is codominance:

a_1a_1 --> A

a_1a_0 --> A,B

a_0a_0 --> B

Lack of dominance and codominance produce three unique phenotypic values for the three possible genotypes.

One of the advantages to diploidy may derive from the production of heterozygotes superior to either homozygote. This also would require three possible values for the trait in the phenotype.

In a genetic algorithm used for function optimization, the dominance map has a range of {0,1}. The two alleles present at homologous loci must always map to either 0 or 1. Therefore codominance and heterozygote superiority cannot be investigated using a genetic algorithm, without major changes in the representation. The two product model can be simulated, however, if there is some control of gene expression.

It has been observed that chromosome segments sometimes become heterochromatic, meaning they condense and appear very dark when stained and viewed under a light microscope. In this state, the segments seem to form no gene products. In mammalian females, one of the two X (sex) chromosomes, apparently selected at random, undergoes heterochromatization (Hood, Wilson, and Wood, 1975). The entire chromosome becomes inactive, leaving the second chromosome to provide any products from the X chromosome genes.

The phenomenon of heterochromatization is modelled by dominance of a random chromosome. One chromosome is selected at random and used to determine the phenotype, as if the second chromosome did not exist.

There is no reason to suppose that heterochromatization is at all related to chromosome fitness. However, it is tempting to consider what the performance of a genetic algorithm would be if in all cases the chromosome with higher fitness value dominated. Dominance of the better chromosome will be included in dominance comparisons, even though it has not been observed in natural systems.

The goal of a dominance scheme in a genetic algorithm is to protect inferior alleles by making them recessive. The construction of a map which benefits an organism in this manner may best be achieved by allowing the genetic algorithm to develop dominance maps dynamically. Each individual in the population carries a third chromosome which acts during evaluation as the dominance map for that individual. During the reproductive cycle, this chromosome behaves like a haploid organism, recombining with the dominance chromosome of the second parent during mating. It mutates with the same frequency as the homologous chromosomes. Offspring creation for organisms with individual dominance maps is illustrated in Figure 5.2. Good dominance maps should develop in parallel with good organisms.

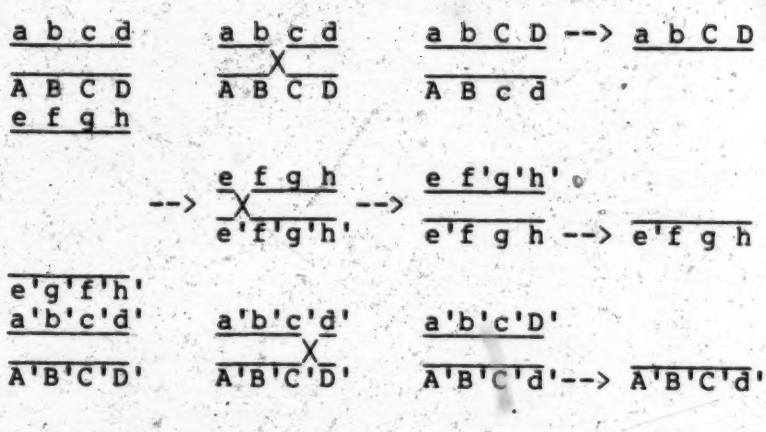


Figure 5.2. Offspring Creation for Diploids with Individual Dominance Maps

The study of possible models for dominance has revealed six dominance schemes which may be useful for genetic algorithms:

fixed, global map,

stochastic, variable, global map,

deterministic, variable, global map,

random chromosome,

better chromosome,

individual maps.

These will be compared in the next section.

5.3 COMPARISONS

5.3.1 Experiment 5: Dominance Schemes

In Experiment 5, the six dominance schemes were run on Test Function Set I with crossover probabilities of 0.3 and 0.5. The factors were

1. Function (1, 2, 4, 5, 6, 7)
2. Dominance Scheme (random chromosome, better chromosome, fixed map, stochastic map, deterministic map, individual maps)
3. Replications (4)
4. Crossover Rate (0.3, 0.5).

The overall best performance is shown in Table 5.1. A crossover rate of 0.5 is superior to 0.3 two thirds of the time. A smaller rate is preferable for a stochastic, variable, global dominance map, which leads to the crossover rate/dominance scheme interaction.

The interaction between dominance schemes and functions is difficult to analyze. Random chromosome dominance performed well on the harder functions, especially Function 7, while stochastic dominance maps did well on the easier functions, 1 and 4. Better chromosome dominance displayed poor performance on the easy functions. The other schemes had similar performances on all functions.

Average Maximum Frequencies are shown in Table 5.2. Here a crossover rate of 0.5 was clearly superior to 0.3. The differences were especially noticeable on Function 6.

Table 5.1. Experiment 5: Dominance Schemes.
Overall Best Performance after 8000 Function Evaluations
(results averaged over 4 runs)

Function	P_c	Dominance Scheme[a]					
		R	B	F	S	D	I
1	.3	95.50	91.00	95.50	98.50	98.50	100.00
	.5	98.50	92.50	100.00	97.00	100.00	100.00
2	.3	66.25	63.75	52.50	67.50	67.50	62.50
	.5	67.50	66.25	57.50	65.00	67.50	67.50
4	.3	98.41	97.95	98.74	98.70	98.43	98.99
	.5	98.25	97.72	98.18	99.15	99.02	97.87
5	.3	93.07	97.35	97.36	94.72	99.84	96.77
	.5	97.36	97.36	99.99	99.47	99.85	99.42
6	.3	88.48	94.00	89.63	93.27	95.41	93.65
	.5	95.53	94.07	97.59	93.60	92.95	93.54
7	.3	96.09	99.96	99.93	92.43	91.08	93.20
	.5	99.95	96.18	99.86	90.03	99.15	96.37

a: dominance schemes
R = random chromosome
B = better chromosome
F = fixed, global map
S = stochastic, variable, global map
D = deterministic, variable, global map
I = individual maps

Source of Variation	M.S.	D.F.	F-ratio	Significant
Function	8514.00	5	369.00	*
Dominance Scheme	24.80	5	1.07	
Crossover Rate	169.70	1	15.00	*
Function x Dom. Scheme	63.80	25	2.76	*
Function x Cross. Rate	9.91	5	.88	
Dom. Scheme x Cross. Rate	26.26	5	2.32	*
Function x Dom. Scheme x Cross. Rate	16.90	25	1.49	
Within Cell	23.08	108		
Cross. Rate x Within Cell	11.31	108		

Table 5.2. Experiment 5: Dominance Schemes.
Average Maximum Frequency after 8000 Function Evaluations
(results averaged over 4 runs)

Function	P_c	Dominance Scheme[a]					
		R	B	F	S	D	I
1	.3	.837	.798	.827	.748	.791	.785
	.5	.842	.788	.844	.749	.779	.772
2	.3	.895	.846	.836	.789	.839	.848
	.5	.887	.834	.820	.774	.841	.836
4	.3	.765	.758	.768	.737	.752	.734
	.5	.735	.727	.751	.731	.755	.732
5	.3	.758	.686	.771	.739	.739	.790
	.5	.759	.685	.741	.727	.736	.775
6	.3	.878	.784	.826	.841	.831	.788
	.5	.776	.726	.784	.815	.704	.710
7	.3	.794	.722	.746	.749	.737	.732
	.5	.791	.714	.726	.725	.745	.754

a: dominance schemes
- R = random chromosome
- B = better chromosome
- F = fixed, global map
- S = stochastic, variable, global map
- D = deterministic, variable, global map
- I = individual maps

Source of Variation	M.S.	D.F.	F-ratio	Significant
Function	.07078	5	43.10	*
Dominance Scheme	.01889	5	11.50	*
Crossover Rate	.02540	1	25.00	*
Function x Dom. Scheme	.00514	25	3.13	*
Function x Cross. Rate	.00845	5	8.32	*
Dom. Scheme x Cross. Rate	.00015	5	.15	
Function x Dom. Scheme x Cross. Rate	.00089	25	.88	
Within Cell	.00164	108		
Cross. Rate x Within Cell	.00102	108		

Better chromosome dominance and stochastic maps provided the greatest population variance. Random chromosome was worst in this respect. The selection of inferior alleles as recessives did preserve population variance significantly better than randomly chosen recessive alleles.

The dynamic development of dominance maps, illustrated by the individual maps dominance scheme, was not particularly effective. Cavicchio (1970) believed that inversion was not useful in genetic algorithms because the population sizes and run lengths were too small for such a subtle operator to show any effect. The dynamic development of dominance maps may be similar to inversion in that respect. The performance of the individual dominance maps scheme was on a par with the deterministic, variable, global dominance maps scheme, but required 50% more space for the dominance chromosomes.

To summarize, the additional space required for individual dominance maps is not justified in view of the unexceptional performance of the individual maps scheme. Random chromosome dominance performs unevenly and has poor variance properties. Better chromosome dominance is ineffective on easy functions. The stochastic map scheme does poorly on hard functions. Fixed, global dominance maps and deterministic, variable, global dominance maps yield even performance and medium levels of population variance relative to the other dominance schemes. Only these two

schemes will be considered further.

5.3.2 Experiment 6: Dominance Change Operators

Holland (1975) believed that diploid organisms would prove most useful to genetic algorithms if there was an operator for dominance inversion. Those rare alleles which had been masked as recessives would occasionally become dominant and undergo evaluation in the environment. One possible dominance change operator for a fixed, global dominance map would require the generation of a new, random map. Dominance change could also be defined as inversion of the current map. A fixed, global map would be replaced by a map with all 0's changed to 1's and vice versa. For a variable, global map, dominance inversion would consist of assigning dominance to the less frequent allele in the previous generation.

Experiment 6 tested the fixed map and deterministic, variable map dominance schemes against the same schemes with periodic dominance changes. For the fixed map scheme, a new random map was generated every 2000 function evaluations. For the deterministic, variable map scheme, dominance inversion was employed. Dominance switched to the less frequent allele for the period between 2000 and 3000 function evaluations and that between 5000 and 6000 function evaluations. The crossover rate for all algorithms was 0.5.

The factors for the experiment were:

1. Function (1, 2, 4, 5, 6, 7)
2. Replications (6)
3. Dominance Scheme (fixed map, deterministic map)
4. Dominance Change Operator (off, on)

The overall best performance measures, shown in Table 5.3, require little explanation. The dominance change operators made a significant difference, often improving algorithm performance. The variances of the populations in the experiment came as a surprise (Table 5.4). The use of dominance change operators sometimes produced a loss of variance. An examination of the behavior of the algorithms over time helps to explain why.

Average Maximum Frequencies for the deterministic map scheme with and without dominance change operators are plotted in Figures 5.3a and 5.3b. Overall best performance for the same algorithms are shown in Figures 5.4a and 5.4b. These are similar to the plots for the fixed map dominance scheme.

Assume that after 2000 function evaluations, the recessives at each locus were mostly the less fit alleles. Reversing the dominance at that point would have subjected these poorer alleles to frequent testing in the environment, resulting in a rapid reduction in their numbers. At the same time, the dominance reversal would have created many new phenotypes. This would have yielded some improvement in algorithm performance. This is what can be observed on the

Table 5.3. Experiment 6: Dominance Change Operators.
Overall Best Performance after 8000 Function Evaluations
(results averaged over 6 runs)

Function	Dominance Change Operator	Dominance Scheme[a]	
		F	D
1	off	97.00	95.00
	on	98.00	100.00
2	off	61.50	69.00
	on	65.00	66.67
4	off	96.79	97.41
	on	98.10	98.47
5	off	97.23	98.88
	on	98.19	100.00
6	off	94.14	92.36
	on	91.47	92.75
7	off	93.02	93.13
	on	98.66	98.12

a: dominance schemes
F = fixed, global map
D = deterministic, variable, global map

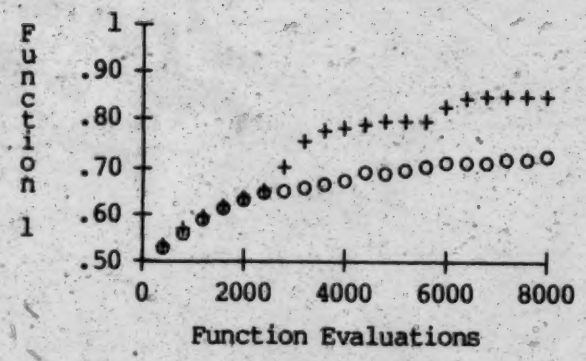
Source of Variation	M.S.	D.F.	F-ratio	Significant
Function	3923.00	5	19.80	*
Dominance Scheme	40.17	1	1.51	
Dom. Change Operator	99.64	1	6.05	*
Function x Scheme	21.20	5	.80	
Function x Operator	29.75	5	1.81	
Scheme x Operator	.06	1	.00	
Function x Scheme x Operator	17.95	5	1.09	
Within Cell	198.00	30		
Scheme x Within Cell	26.54	30		
Operator x Within Cell	16.47	30		
Scheme x Operator x Within Cell	16.40	30		

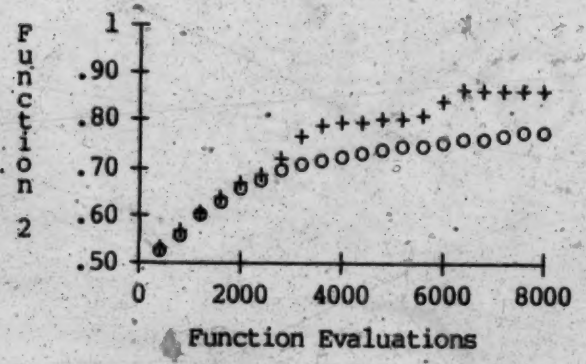
Table 5.4. Experiment 6: Dominance Change Operators.
Average Maximum Frequency after 8000 Function Evaluations
(results averaged over 6 runs)

Function	Dominance Change Operator	Dominance Scheme[a]	
		F	D
1	off	.822	.730
	on	.849	.857
2	off	.821	.782
	on	.874	.866
4	off	.769	.727
	on	.775	.766
5	off	.772	.717
	on	.784	.774
6	off	.788	.822
	on	.731	.830
7	off	.750	.724
	on	.736	.741

a: dominance schemes
F = fixed, global map
D = deterministic, variable, global map

Source of Variation	M.S.	D.F.	F-ratio	Significant
Function	.03316	5	13.72	*
Dominance Scheme	.00462	1	3.99	
Dom. Change Operator	.03230	1	22.60	*
Function x Scheme	.00940	5	8.12	*
Function x Operator	.00904	5	6.33	*
Operator x Scheme	.02335	1	46.30	*
Function x Scheme x Operator	.00117	5	2.31	
Within Cell	.00242	30		
Scheme x Within Cell	.00116	30		
Operator x Within Cell	.00143	30		
Scheme x Operator x Within Cell	.00050	30		





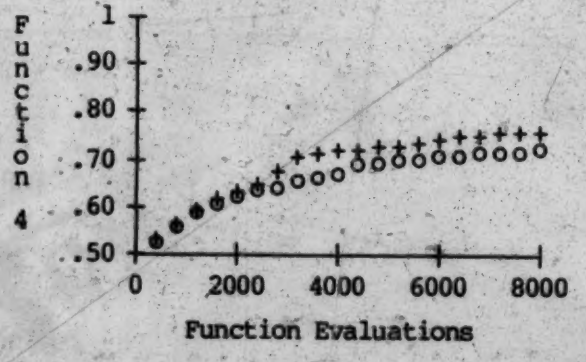
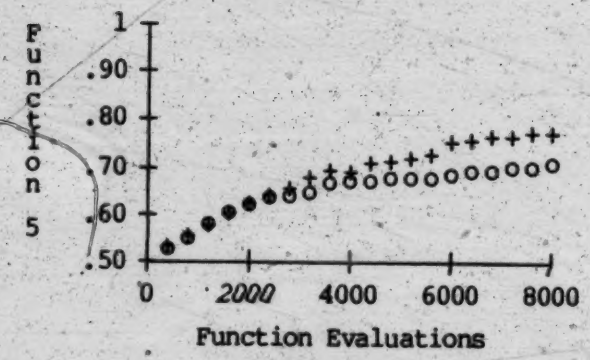
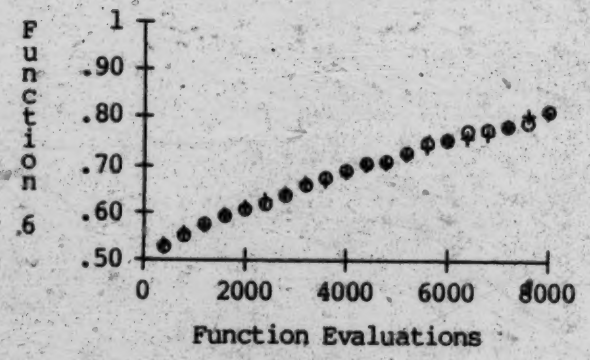


Figure 5.3a. Experiment 6: Dominance Change Operators.
Average Maximum Frequency, Dominance = Deterministic Map
(o) Dominance Change Operator Off
(+) Dominance Change Operator On





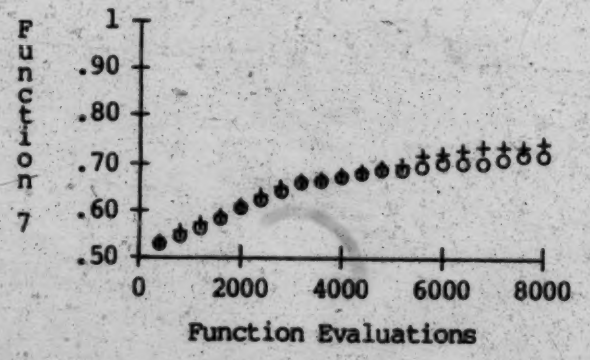
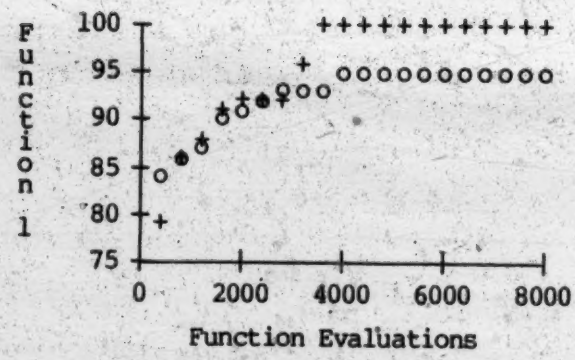
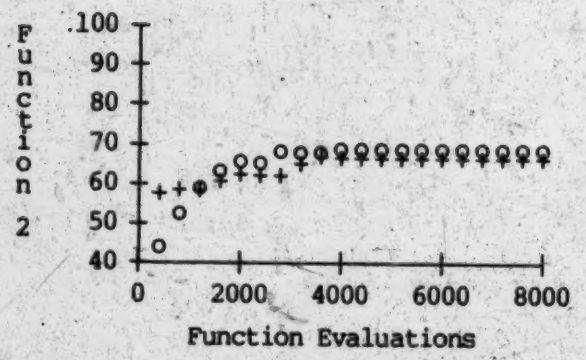


Figure 5.3b. Experiment 6: Dominance Change Operators.
Average Maximum Frequency, Dominance = Deterministic Map
(o) Dominance Change Operator Off
(+) Dominance Change Operator On





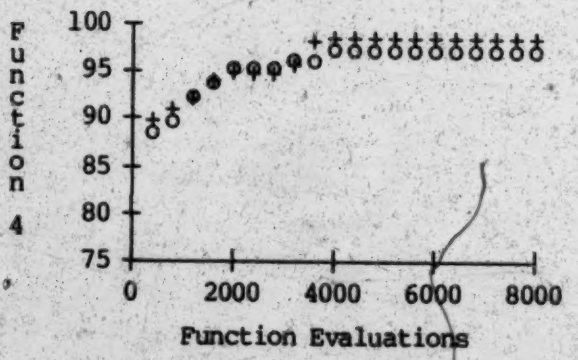


Figure 5.4a. Experiment 6: Dominance Change Operators.
Overall Best Performance, Dominance = Deterministic Map
(o) Dominance Change Operator Off
(+) Dominance Change Operator On

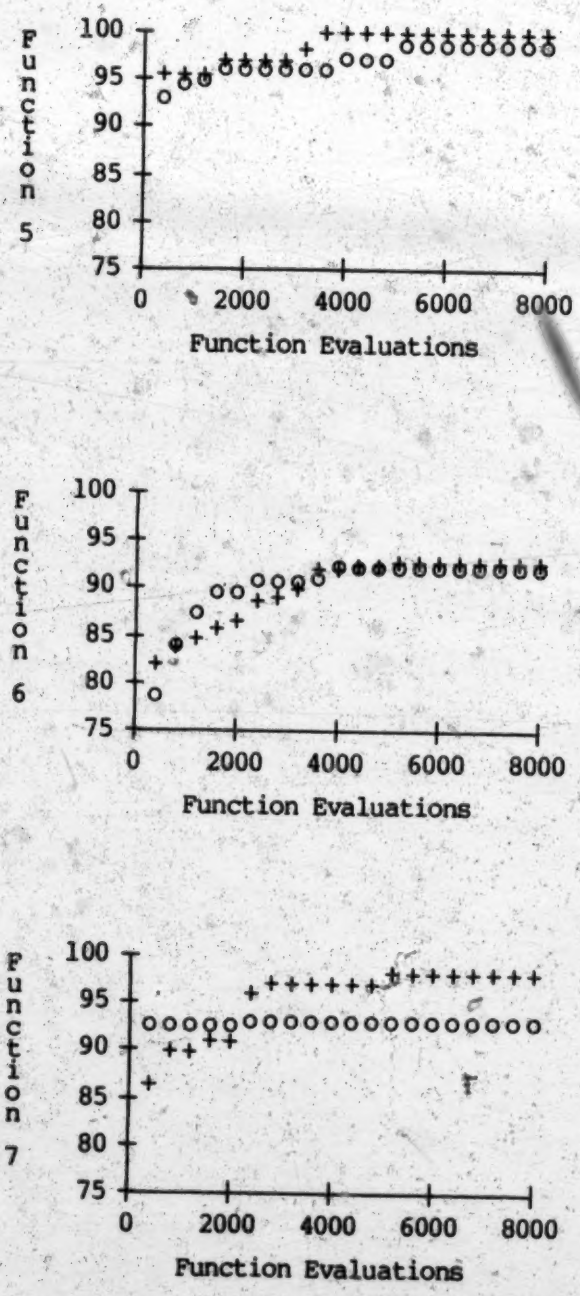


Figure 5.4b. Experiment 6: Dominance Change Operators.
Overall Best Performance, Dominance = Deterministic Map
(o) Dominance Change Operator Off
(+) Dominance Change Operator On

easier functions, especially Function 1.

Now assume that there was no initial advantage to either allele at many of the loci, so that dominance drifted randomly to one allele or the other. Reversing the dominance after 2000 function evaluations would have permitted many marginally better (but previously recessive) alleles to increase their numbers in the population. The result would have been immediate improvements in performance and very little loss of variance. This is what can be observed on the harder functions, most notably on Function 7.

5.3.3 Experiment 7: Haploidy vs. Diploidy

Experiment 7 was designed to compare the performance of haploid and diploid algorithms on function optimization problems. Haploidy was represented by an algorithm with population 200, mutation rate 0.001, and crossover rate 0.6. An algorithm using the deterministic map dominance scheme with dominance change operator was selected as a reasonable representative of the diploid algorithms. Population size was 100, so that space requirements for the two algorithms, measured as number of chromosomes in the population, were equal. The mutation rate for the diploid algorithm was 0.000025; the crossover rate was 0.5. Both algorithms used deterministic parent selection and the all-time worst value as the base of the selection distribution.

As before, runs were terminated after 8000 function evaluations. Thus the haploid algorithm ran for 40 generations, the diploid algorithm for 80. Factors in the experiment were

1. Function (1, 2, 4, 5, 6, 7)
2. Algorithm (haploid, diploid)
3. Replications (10)

Table 5.5 gives performance results for Experiment 7. Diploidy did not improve overall best performance. Best performance was plotted over time (Figures 5.5a and 5.5b), to determine whether the diploid algorithm had had time to fulfill its potential for search. It appears that on all functions, both algorithms had reached plateaus of performance by 7000 function evaluations.

Average Maximum Frequencies for the two algorithms are shown in Table 5.6. There was a small function/algorithm interaction, but in general diploidy with the dominance change operator did not increase population variance.

5.3.4 Experiment 8: Diploidy for Limited Resources

The good early performance of the diploid algorithm in Experiment 7 suggests another possible application of diploidy: to problems with severe limitations on time and space. Experiment 8 compared a diploid and a haploid algorithm over Test Function Set I for runs of only 2000 function evaluations. Population sizes for both algorithms were limited to 20 individuals. The early performance of the diploid algorithm in Experiment 7 did not depend on the

Table 5.5. Experiment 7: Haploidy vs. Diploidy.
Overall Best Performance after 8000 Function Evaluations
(results averaged over 10 runs)

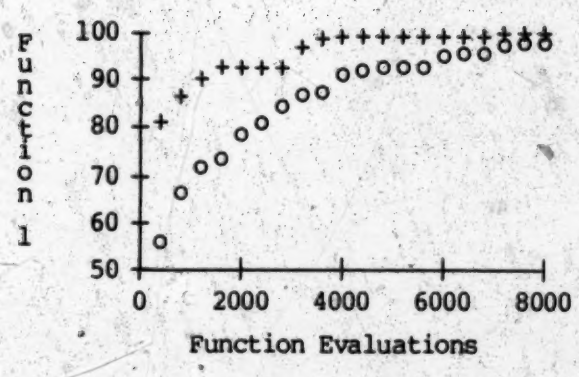
Algorithm: / Function:	Haploid	Diploid
1	98.80	100.00
2	68.00	65.00
4	98.94	97.35
5	99.97	99.94
6	93.62	92.65
7	99.12	97.86

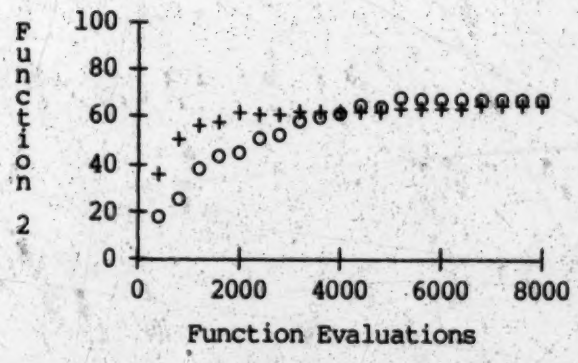
algorithms:

Haploid: parent selection = deterministic
population size = 200
crossover rate = 0.6
mutation rate = 0.001
ploidy = 1

Diploid: parent selection = deterministic
population size = 100
crossover rate = 0.5
mutation rate = 0.000025
ploidy = 2
dominance scheme = deterministic map
dominance change operator = on

Source of Variation	M.S.	D.F.	F-ratio	Significant
Function	3389.00	5	441.00	*
Algorithm	26.65	1	3.47	
Function x Algorithm	10.17	5	1.32	
Within Cell	7.69	108		





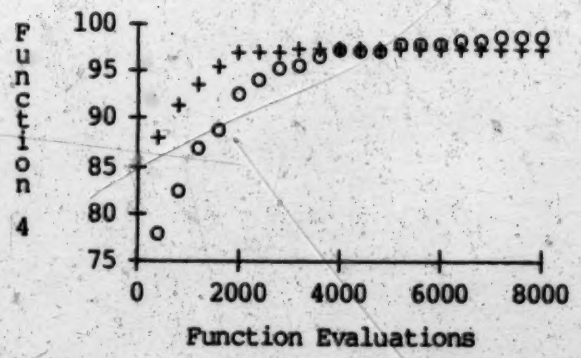


Figure 5.5a. Experiment 7: Haploidy vs. Diploidy.
Overall Best Performance on Functions 1, 2, and 4
(o) Haploid Algorithm
(+) Diploid Algorithm

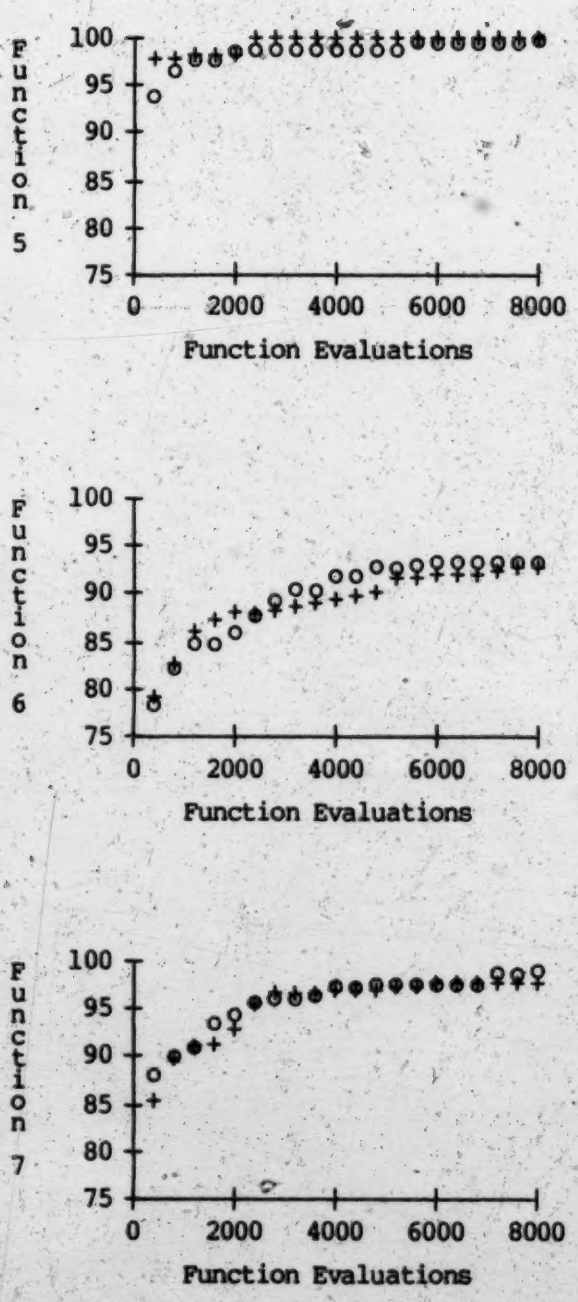


Figure 5.5b. Experiment 7: Haploidy vs. Diploidy.
Overall Best Performance on Functions 5, 6, and 7
(o) Haploid Algorithm
(+) Diploid Algorithm

Table 5.6. Experiment 7: Haploidy vs. Diploidy.
Average Maximum Frequency after 8000 Function Evaluations
(results averaged over 10 runs)

Algorithm: / Function:	Haploid	Diploid
1	.832	.840
2	.890	.852
4	.730	.760
5	.720	.751
6	.763	.780
7	.703	.733

algorithms:

Haploid: parent selection = deterministic
population size = 200
crossover rate = 0.6
mutation rate = 0.001
ploidy = 1

Diploid: parent selection = deterministic
population size = 100
crossover rate = 0.5
mutation rate = 0.000025
ploidy = 2
dominance scheme = deterministic map
dominance change operator = on

Source of Variation	M.S.	D.F.	F-ratio	Significant
Function	.07431	5	54.00	*
Algorithm	.00520	1	3.78	
Function x Algorithm	.00348	5	2.53	*
Within Cell	.00138	108		

dominance change operator, so for Experiment 8 the diploid algorithm used the deterministic map dominance scheme with no dominance change operator. Other algorithm parameters remained unaltered from Experiment 7. The factors were:

1. Function (1, 2, 4, 5, 6, 7)
2. Algorithm (haploid, diploid)
3. Replications (10)

Overall best performance after 2000 function evaluations is given in Table 5.7. Plots of overall best are shown in Figures 5.6a and 5.6b. The diploid algorithm had a small advantage on Function 5. In general, however, there was no significant difference between haploidy and diploidy, despite the fact that the diploid algorithm was allowed twice as many chromosomes in its population as the haploid algorithm.

5.4 DIPLOIDY IN RETROSPECT

The failure of diploidy to produce significant improvements in genetic algorithms for function optimization can be better understood in light of current theories in population genetics concerning diploidy. Diploidy and recombination have been popular subjects in the literature, beginning with Fisher in 1958 and continuing to the present. (See Maynard Smith, 1978, Crow and Kimura, 1970, Ewens, 1968, and Fisher, 1958, for general treatments. Li, 1978, and Felsenstein, 1974, are representative of research papers on recombination.) In his book, The Evolution of Sex (1978), Maynard Smith makes several relevant points.

Table 5.7. Experiment 8: Diploidy for Limited Resources.
Overall Best Performance after 2000 Function Evaluations
(results averaged over 10 runs)

Algorithm: / Function:	Haploid	Diploid
1	73.00	74.40
2	43.50	36.00
4	87.23	88.16
5	85.40	90.12
6	81.55	84.29
7	69.55	70.71

algorithms:

Haploid: parent selection = deterministic
population size = 20
crossover rate = 0.6
mutation rate = 0.001
ploidy = 1

Diploid: parent selection = deterministic
population size = 20
crossover rate = 0.5
mutation rate = 0.000025
ploidy = 2
dominance scheme = deterministic map
dominance change operator = off

Source of Variation	M.S.	D.F.	F-ratio	Significant
Function	6577.00	5	39.43	*
Algorithm	16.44	1	.10	
Function x Algorithm	90.64	5	.54	
Within Cell	166.80	108		

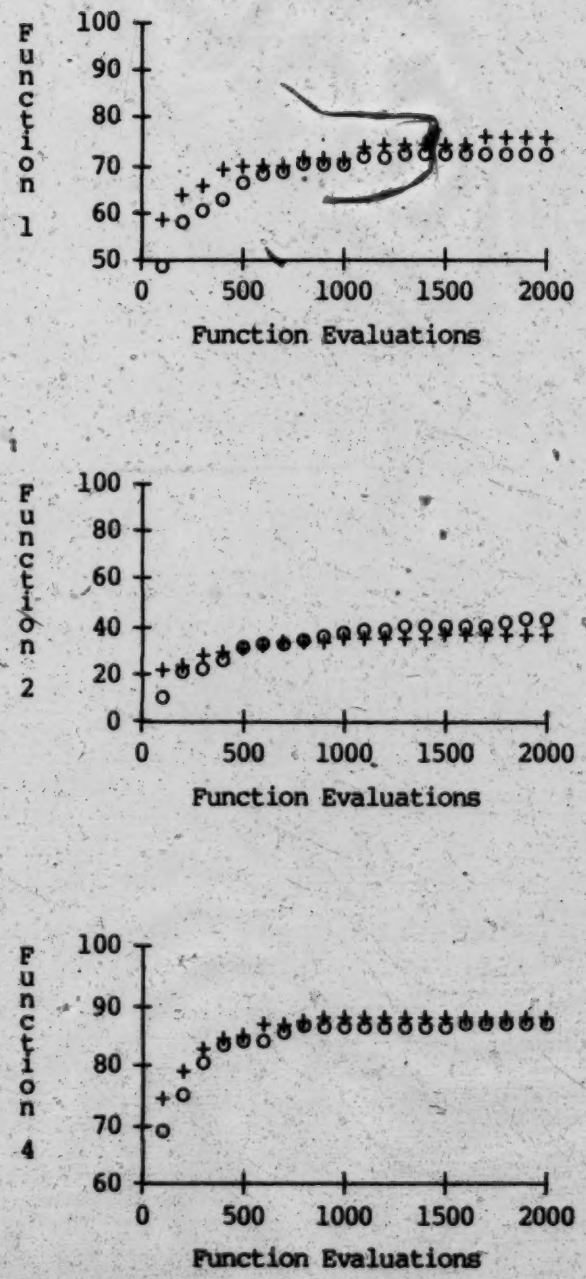
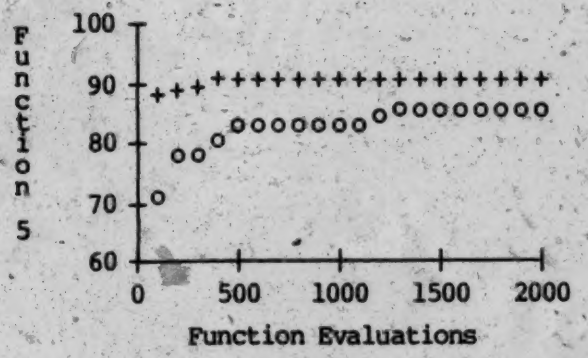
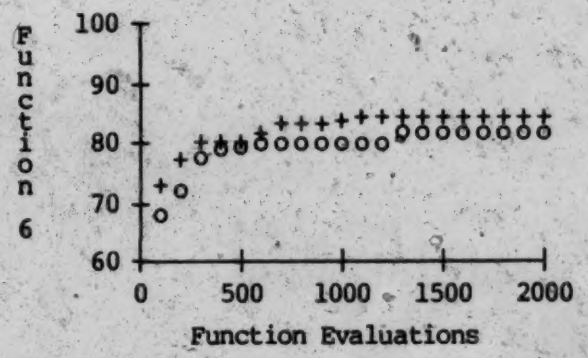


Figure 5.6a. Experiment 8: Diploidy for Limited Resources
Overall Best Performance on Functions 1, 2, and 4
(o) Haploid Algorithm
(+) Diploid Algorithm





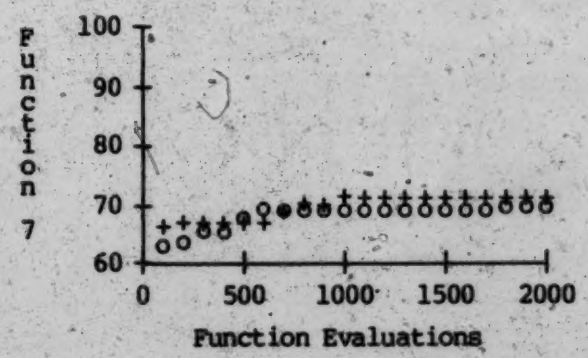


Figure 5.6b. Experiment 8: Diploidy for Limited Resources
Overall Best Performance on Functions 5, 6, and 7
(o) Haploid Algorithm
(+) Diploid Algorithm

First, it is possible that diploidy is beneficial during somatic development (growth of the organism after fertilization), providing protection against deleterious mutations. In this role, diploidy would aid in preserving the viability of individual organisms, but would not affect the adaptive evolution of the population.

Second, the purpose of recombination may be to provide more rapid evolution of populations. The conditions found by Maynard Smith under which recombination is beneficial are listed below.

Heterozygote Superiority. Recombination of diploid organisms is desirable when the heterozygote at a locus is superior to either homozygote. This situation does not develop in genetic algorithms for function optimization which use binary representations.

Dependent Loci. The fitness of the combination of alleles A and B at two loci may be superior to that of A or B alone, that is, $f(AB) > f(A) \cdot f(B)$. Then if $p(AB,t) < p(A,t) \cdot p(B,t)$, recombination will increase the number of AB individuals in the population, even if $p(AB,t)$ is initially zero. Thus recombination benefits the search for good schemata when the alleles defining the schemata are already in the population. Recombination and variance in the population together provide better adaptive search.

Missing Optimal Alleles. When the optimal alleles at several loci are missing from a population, they must be introduced by mutation. This condition could be caused by

shifts in the environment which render a previously inferior allele optimal. If the population does not recombine, mutations must all occur over several generations in the same family line in order to produce at least one individual containing all the mutations.

Recombination allows mutations in two or more individuals of the same generation to come together in one individual within only a few time steps. So if $nP_m > 1$, that is, more than one mutation occurs in one generation, recombination will reduce the time needed to produce individuals with all of the optimal alleles. Recombination in this situation increases the speed with which mutation can compensate for the absence of desirable alleles.

From these cases, it becomes clear that the rate of response of a population to environmental factors depends not on haploidy and diploidy per se, but on

1) the variance of the population, especially whether or not alleles are totally absent from the population, and
2) whether or not there is recombination between individuals in the population.

Diploidy in natural systems may be useful in somatic development as a protection against harmful mutations. This use does not apply genetic algorithms. The use of a diploid reproductive cycle may improve population fitness and response to the environment under conditions of heterozygote superiority, dependent loci, and missing optimal alleles. The keys to this improved response are the maintenance of

population variance and the use of recombination.

It has been shown that in a genetic algorithm, a population size of 200 and a mutation rate of 0.001 are sufficient to prevent fixation. Crossover provides a powerful tool for recombination. It is apparent after the experiments of this chapter that diploidy cannot provide more population variance or recombination in a genetic algorithm than has already been obtained by haploid algorithms employing the crossover and mutation operators. However, it must be kept in mind that this result applies only to genetic algorithms with binary representations used on function optimization problems.

Diploidy and dominance are not well understood. The question "why diploidy?" has not yet been answered for natural systems. But if the utility of diploidy does derive primarily from somatic differentiation, heterozygote superiority, recombination, and maintenance of population variance, then the benefits of diploidy do not transfer to genetic algorithms for function optimization.

5.5 CONCLUSIONS

Under diploidy, it is possible to reduce crossover and mutation rates without reducing levels of recombination or population variance. However, with respect to overall best performance on difficult function optimization problems, diploidy does not improve genetic algorithms.

Diploidy may be useful for dynamic environments, in which case a dominance scheme which develops dynamically to reflect allele fitness is good. Deterministic, variable, global dominance maps are an example of such a scheme. Dominance change operators can improve overall best performance on diploid populations, especially on difficult functions.

CHAPTER 6

COMPARISON OF GLOBAL FUNCTION OPTIMIZATION METHODS

Many practical problems in global function optimization are characterized by *a priori* ignorance. The number of dimensions and bounds on each dimension are usually known, but there is no information about the behavior of the function. Such "black box" functions require an optimization algorithm which makes few assumptions about the nature of the function and works well over a wide range of functions.

Genetic algorithms optimize bounded functions over any number of dimensions. They do not require derivatives and are not limited to functions which are unimodal in Euclidean space. Logically, the best area in which to apply genetic algorithms in optimization of Euclidean functions is the area of global "black box" optimization. This chapter investigates that application.

6.1 NUMERICAL METHODS

Numerical methods for global optimization can be categorized according to what assumptions they make about the function. Methods which are tailored to a class of functions will be superior to general methods on those functions, but their performance elsewhere may not be predictable. Therefore, for "black box" functions it is

desirable to minimize the number of assumptions made. The following is a catalog of the best numerical methods currently available which do not require continuity or differentiability of the function.

No Assumptions. The only methods which make no assumptions whatsoever about the function are random search methods. Pure random search consists of the evaluation of a given number of points chosen at random from the solution space. Modifications to pure random search such as Anderssen's (1972) are also unrestricted in application, but have not yet been well analyzed.

Grouped Optimal Points. Several methods assume only that optimal points are grouped in neighborhoods in Euclidean space. If this is the case, searching should be intensified near previously located good points. Creeping random search as proposed by Brooks (1958) centers a set of normally distributed trials around an initial point, then moves the search to center around the best point found before repeating the process. Clustering algorithms have also been employed to locate optimal neighborhoods after an initial set of random trials has been completed (Price, 1977).

Smoothness. If the function is assumed to be reasonably smooth in Euclidean space, a grid can be imposed on the solution space such that each grid sector will probably contain at most one relative optimum. These sectors can then be searched by evaluating sector centroids

(Brooks, 1958) or by using some method for finding local optima (e.g. Hill, 1969). The difficulty with such grid methods lies in the determination of an initial sector size which corresponds to the smoothness of the function. Failure to adopt the appropriate initial grid is usually disastrous.

Quadratic Smoothness. If a function behaves much like a quadratic polynomial in the neighborhoods of local optima, it can be searched by local direct search or descent methods. Jarvis (1975), Jacoby, Kowalik, and Pizzo (1972), Kowalik and Osborne (1968), and Fletcher (1965; 1969) provide reviews of these methods. A global optimum can be located by running these local methods in global mode, i.e. applying them repeatedly from different starting points.

On one-dimensional functions, search by quadratic approximation is often used. For higher numbers of dimensions, the best direct search methods are Rosenbrock's method as modified by Davies, Swann, and Campey (Box, Davies, and Swann, 1969; Swann, 1964), the Simplex method of Nelder and Mead (1965), and Powell's method (Powell, 1964; Brent, 1972). Gradient methods, which use derivatives to find the direction of greatest change in the function, can often be altered so that the derivatives are approximated. Davidon's variable metric method as modified by Stewart (Fletcher and Powell, 1963; Stewart, 1967) is a good example.

The Stewart and Powell methods are probably the most powerful local methods in terms of speed of convergence in the later stages of search. However, the method of Davies, Swann, and Campey exhibits good early performance. It also may be preferable for functions with large numbers of dimensions and may perform better than other quadratic methods on functions which do not satisfy assumptions of quadratic smoothness (Fletcher, 1965).

Hamming Space Linear Independence. Genetic algorithms do not require smoothness of the function in Euclidean space. Instead, their performance is related to the linearity of the function when transformed into Hamming space.

In summary, the only methods which make no assumptions about the function are random search methods. Other methods may be valid on "black box" functions, however, if their performance is sufficiently robust. The most likely candidates for global optimization in environments of *a priori* ignorance are pure random search (PRS), creeping random search (CRS), the Davies, Swann, and Campey method used in global mode (DSC), and the genetic algorithm (GA). These will be compared in Section 6.3.

6.2 TEST FUNCTION SET II: ALGORITHM COMPARISON

In order to compare the four global optimization methods selected in Section 6.1, it is necessary to have a set of test functions which will challenge the robustness of each method. In addition, all the functions must be bounded

and have two or more dimensions, so that all four methods are applicable. The function characteristics which should be varied within the test set are smoothness in Euclidean space, Euclidean modality, and Hamming space linear dependencies. Also, DSC may prove sensitive to the number of dimensions in the function, so this should be varied as well.

The following ten functions comprise Test Function Set II. Three functions were taken from the literature. However, few experiments have been published which use highly multimodal functions, so functions 3 through 8 and 10 were constructed to provide the desired variations in modality. The bounds for most functions were set so that the genetic algorithm could search spaces with a resolution for each x-value of 10^{-10} and a space size of some power of two. The main features of each function are given in Table 6.1.

6.2.1 Function 1

$$f1(x_1,x_2) = 100 - 100(x_1^2 - x_2)^2 - (1 - x_1)^2.$$

Bounds: $-3.4359738368 \leq x_i \leq 3.4359738367$,

for i = 1 and 2. $3.4359738368 = 2^{35} \cdot 10^{-10}$.

Maximum: $f1(1,1) = 100.0$

Function 1 is Rosenbrock's valley (Rosenbrock, 1960), negated to form a maximization problem. It is a quartic function which is unimodal in Euclidean space, but not strongly unimodal. It is representative of the unimodal functions which are used to compare direct search and

Table 6.1. Test Function Set II.

Function	Number of Dimensions	Euclidean Modality	Hamming Linearity	Continuous?
1. Rosenbrock's Valley	2	1	Nonlinear	Yes
2. Shekel's Foxholes	2	25	Nonlinear	Yes
3. 2-D Fourier Sum	2	10^8	Nonlinear	Yes
4. Damped Sine	2	64	Nonlinear	Yes
5. 10-D Fourier Sum A	10	59049	Nonlinear	Yes
6. 10-D Fourier Sum B	10	59049	Linearly Independent	Yes
7. Product of Fourier Pairs	10	?	Nonlinear	Yes
8. Full Fourier Product	10	?	Nonlinear	Yes
9. Step Function	10	1	Linearly Independent	No
10. Hamming Function	5	$5.6 \cdot 10^{14}$	Linearly Independent	Yes

descent methods in the literature.

6.2.2 Function 2

Let $A = (a_1,\ldots,a_{25}) = (-32, -16, 0, 16, 32,$
$-32, -16, 0, 16, 32,$
$-32, -16, 0, 16, 32,$
$-32, -16, 0, 16, 32,$
$-32, -16, 0, 16, 32)$, and
$B = (b_1,\ldots,b_{25}) = (-32, -32, -32, -32, -32,$
$-16, -16, -16, -16, -16,$
$0, 0, 0, 0, 0,$
$16, 16, 16, 16, 16,$
$32, 32, 32, 32, 32).$

$$f2(x_1,x_2) = 100.998 - \{.002 + \sum_{i=1}^{25} [i+(x_1-a_i)^6+(x_2-b_i)^6]^{-1}\}^{-1}.$$

Bounds: $-54.975581388 \leq x_i \leq 54.975581387,$
for $i = 1$ and 2. $54.975581388 = 2^{39}\cdot 10^{-10}$.

Maximum: $f2(-32,-32) = 100.0$

Function 2 is Shekel's foxholes function as used by De Jong (De Jong, 1975; Shekel, 1971). It also has been negated to form a maximization problem. The surface is fairly flat with a minimum value of about -400. There are 25 steep peaks at the points (a_i,b_i), $i=1,\ldots,25$, each with a relative maximum value of about $101 - i$.

6.2.3 Function 3

$$f3(x_1,x_2) = \sum_{i=1}^{2} \sum_{j=1}^{5} 10\sin(\pi a_{ij}(x_i + 9b_{ij})),$$

where for j=1,...,5,

a_{1j} = 18818, 235298, 1940596, 21112002, 200040002,

a_{2j} = 21218, 207646, 2060602, 19531250, 199720098,

$b_{1j} = -.25858495 \cdot 10^{-4}, .69532345 \cdot 10^{-6}, -.64058648 \cdot 10^{-6},$
$-.17853847 \cdot 10^{-6}, -.67849272 \cdot 10^{-8},$

$b_{2j} = -.28863902 \cdot 10^{-4}, -.18619699 \cdot 10^{-5}, .22571332 \cdot 10^{-6},$
$.256 \cdot 10^{-7}, -.2924699 \cdot 10^{-8}.$

Bounds: $0 \leq x_i \leq .0001048575$, for i=1,2.

$.0001048576 = 2^{20} \cdot 10^{-10}$.

Maximum: $f3(2^{19} \cdot 10^{-10}, 2^{19} \cdot 10^{-10}) = 100.0$

Function 3 is the sum over two dimensions of Fourier sums which are similar to Function 6 in Test Function Set I (Section 3.4.6). It contains roughly 10^8 relative maxima. The periods of all the sine waves are relatively prime to each other and to 2.

6.2.4 Function 4

Let $y_i = x_i/(2^{30} \cdot 10^{-10})$, for i=1,2.

$$f4(x_1,x_2) = 50 \sum_{i=1}^{2} (1-y_i)^4 \sin(16\pi(y_i + .05874169672)^{-.25}).$$

Bounds: $0 \leq x_i \leq .1073741823$, for i=1,2.

$.1073741824 = 2^{30} \cdot 10^{-10}$, so $0 \leq y_i < 1$.

Maximum: $f4(0,0) = 100.0$

Function 4 is the sum over two dimensions of the dampened sinusoid of Test Function Set I (Section 3.4.7). The function has eight ridges crossing the space in each of the directions of the x_i. It is smooth in Euclidean space, but has very unevenly spaced relative maxima.

6.2.5 Function 5

Let $X = (x_1,\ldots,x_{10})$.

$$f5(X) = 5\sum_{i=1}^{10}\sum_{k=1}^{2}\sin(\pi a_k(x_i + b_k)),$$

where $a_1 = 19531250$, $a_2 = 78125000$, $b_1 = -.256\cdot10^{-7}$, $b_2 = -.64\cdot10^{-8}$.

Bounds: $0 \leq x_i \leq .0000001023$, for i=1,...,10.

$.0000001024 = 2^{10}\cdot10^{-10}$.

Maximum: $f5(X) = 100.0$ for $x_i = 5.12\cdot10^{-8}$, i=1,...,10.

Function 5 is the sum of 10 Fourier sums. Each Fourier sum has two component sine waves, one with one peak and one with three. This gives a smooth, high-dimensional surface with $3^{10} = 59049$ relative maxima evenly distributed over the space.

6.2.6 Function 6

Let $X = (x_1,\ldots,x_{10})$.

$$f6(X) = 5\sum_{i=1}^{10}\sum_{k=1}^{2}\sin(\pi a_k(x_i + b_k)),$$

where $a_1 = 19531250$, $a_2 = 78125000$, $b_1 = -.32\cdot10^{-7}$, $b_2 = 0$.

Bounds: $0 \leq x_i \leq .0000001023$, for i=1,...,10.

Maximum: $f6(X) = 100.0$ for $x_i = 5.76\cdot10^{-8}$, i=1,...,10.

Function 6 is similar to Function 5, but the sine waves for each dimension are shifted slightly so that Hamming space nonlinearity is reduced.

6.2.7 Function 7

Let $X = (x_1,\ldots,x_{10})$.

$$f7(X) = 5\sum_{i=0}^{4}\prod_{j=1}^{2}\sum_{k=1}^{2}\sin(\pi a_k(x_{2i+j} + b_k)),$$

where a_k, b_k are as in Function 5, for k=1,2.

Bounds: $0 \le x_i \le .0000001023$, for i=1,...,10.

Maximum: $f7(X) = 100.0$ for $x_i = 5.12\cdot10^{-8}$, i=1,...,10.

Function 7 is the sum of five products involving the ten Fourier sums used in Function 5. Thus, in contrast to Functions 3, 4, 5, and 6, Function 7 will have some nonlinearity.

6.2.8 Function 8

Let $X = (x_1,\ldots,x_{10})$.

$$f8(X) = .09765625\prod_{i=1}^{10}\sum_{j=1}^{2}\sin(\pi a_j(x_i + b_j)),$$

where a_j, b_j are as in Function 5, for j=1,2.

Bounds: $0 \le x_i \le .0000001023$, for i=1,...,10.

Maximum: $f8(X) = 100.0$ for $x_i = 5.12\cdot10^{-8}$, i=1,...,10.

Function 8 is the product of the ten Fourier sums used in Function 5. It will have a very high level of nonlinearity.

6.2.9 Function 9

Let $X = (x_1,\ldots,x_{10})$.

$$f9(X) = \sum_{i=1}^{5}\lfloor x_i \rfloor + \sum_{i=6}^{10}\lfloor 10.23 - x_i \rfloor.$$

Bounds: $0 \leq x_i \leq 10.23$, for i=1,...,10.

Maximum: f9(X) = 100 when $x_i \geq 10$, i=1,...,5, and $x_i \leq .23$, i=6,...,10.

Function 9 is a discontinuous, high-dimensional step function. It has one relative maximum value, which occurs over a hypercube in one corner of the solution space. The corner chosen has five high x-values and five low ones, to discourage a method which searches all dimensions in the same direction initially.

6.2.10 Function 10

Let $X = (x_1,...,x_5)$.

$$f10(X) = 2\sum_{i=1}^{5} \text{Hamming Distance of } \lfloor x_i \rfloor \text{ to } 512.$$

Bounds: $0 \leq x_i \leq 1023$, for i=1,...,5.

Maximum: f10(X) = 100 when $512 \leq x_i < 513$, i=1,...,5.

Function 10 is a five dimensional version of the Hamming function (Function 1) of Test Function Set I (Section 3.4.1). It is linearly independent in Hamming space and has $2^{49} \approx 5.6 \cdot 10^{14}$ peaks in Euclidean space.

6.3 EXPERIMENT 9: GLOBAL OPTIMIZATION

In order to compare the four methods of Section 6.1, it is necessary to set up similar modes of operation for each method. The commonly used global mode for DSC involves running the method to convergence on each of several random starting points. However, the genetic algorithm is always

allowed an initial population of 200 random starting points. The genetic algorithm should logically be compared to a DSC method which searches in parallel starting from 200 random points. Thus for CRS, DSC, and GA, two running modes will be used:

Convergence mode ("c" mode). Run until convergence starting from one random point (or random population), then restart. Continue up to a maximum of 10000 function evaluations.

Fixed starts mode ("f" mode). Use a fixed number of random starting points. Spend the same amount of time searching from each point, so that the total number of function evaluations is 10000.

The parameters used for each of the methods are described in Appendix D. Seven methods, PRS, CRS_c, CRS_f, DSC_c, DSC_f, GA_c, and GA_f, were run 10 times each on each function in Test Function Set II.

The overall best performances of the methods are given in Table 6.2. Creeping random search surpassed pure random search on only one function, and was worse on several. Since CRS is more expensive than PRS to use, requiring the generation of normally distributed points, it need not be considered further as a global optimization technique.

DSC exhibited marked differences in performance depending on the mode of operation used (Table 6.3). DSC employed only 50 to 100 random starting points when in convergence mode. On the dampened sinusoid (Function 4),

Table 6.2. Experiment 9: Global Optimization.
Overall Best Performance after 10000 Function Evaluations
(results averaged over 10 runs)

Function	Method[a]						
	PRS	CRS(c)	CRS(f)	DSC(c)	DSC(f)	GA(c)	GA(f)
1	99.99	98.44	99.99	100.00	100.00	99.98	99.98
2	99.95	33.04	99.85	99.31	100.00	99.45	99.80
3	78.76	50.19	78.76	90.70	90.82	92.34	91.13
4	82.06	44.84	82.40	77.22	88.79	93.62	92.41
5	57.89	12.54	51.90	86.10	81.23	79.63	81.13
6	59.04	23.46	51.74	93.22	85.38	88.27	92.40
7	- .11	20.00	20.00	19.82	19.97	20.00	20.00
8	3.54	.01	.99	31.79	10.27	2.52	4.50
9	80.30	52.10	72.10	98.90	96.40	93.00	92.50
10	76.80	63.00	75.20	88.40	85.00	93.40	93.40

a: method
PRS = pure random search
CRS = creeping random search
DSC = numerical method of Davies, Swann, and Campey
GA = genetic algorithm
(c) = convergence mode
(f) = fixed starts mode

Source of Variation	M.S.	D.F.	F-ratio	Significant
Method	19200.00	6	103.30	*
Function	66400.00	9	357.30	*
Method x Function	1344.00	54	7.23	*
Within Cell	185.80	630		

Table 6.3. Experiment 9: Global Optimization.
Overall Best Performance of DSC
after 10000 Function Evaluations
(results averaged over 10 runs)

Function	Mode	
	Convergence	Fixed Starts
1	100.00	100.00
2	99.31	100.00
3	90.70	90.82
4	77.22	88.79
5	86.10	81.23
6	93.22	85.38
7	19.82	19.97
8	31.79	10.27
9	98.90	96.40
10	88.40	85.00

Source of Variation	M.S.	D.F.	F-ratio	Significant
Method	380.80	1	27.70	*
Function	18620.00	9	1350.00	*
Method x Function	346.80	9	25.20	*
Within Cell	13.76	180		

using 200 random starting points in fixed starts mode increased the probability of initiating hill-climbing on one of the narrower, higher ridges. On the more regular Fourier sums (Functions 5 and 6), DSC performed well when allowed to converge on a peak after each start.

The genetic algorithm was not affected by operating mode (Table 6.4). The algorithm maintained population variance at a level which prevented convergence within 10000 function evaluations, and the algorithm was rarely restarted on a new, random population. Thus the two modes behaved the same. Because fixed starts mode is the natural way of running the genetic algorithm, only GA_f will be considered in further discussions.

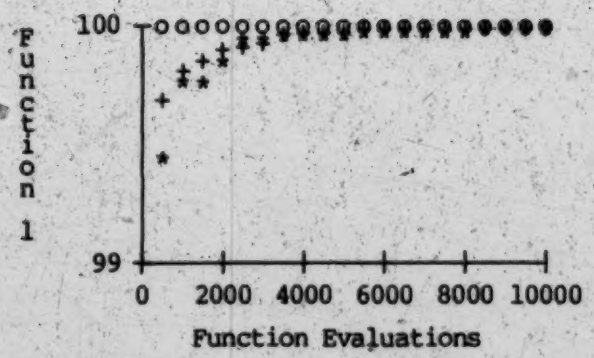
The genetic algorithm always equalled or surpassed pure random search. DSC, however, offered stiff competition. Plots of overall best performance of PRS, DSC_c, and GA_f for all ten functions are given in Figures 6.1a, b, c, and d. Each point is an average of 10 runs. Note that the vertical scale varies from function to function.

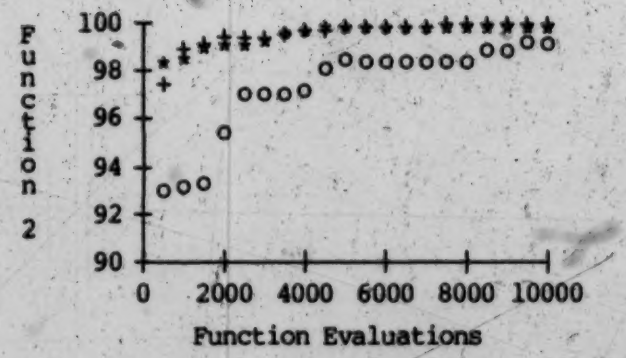
On all functions except the linearly dependent Fourier products (Functions 7 and 8), the genetic algorithm exhibited a steadily improving performance curve. On functions linearly independent in Hamming space (Functions 6, 9, and 10), the plots indicate that useful search was still in progress at the arbitrary cutoff point of 10000 function evaluations. On Functions 7 and 8, however, locus dependencies prevented effective search.

Table 6.4. Experiment 9: Global Optimization.
Overall Best Performance of GA
after 10000 Function Evaluations
(results averaged over 10 runs)

Function	Mode	
	Convergence	Fixed Starts
1	99.98	99.98
2	99.45	99.80
3	92.34	91.13
4	93.62	92.41
5	79.63	81.13
6	88.27	92.40
7	20.00	20.00
8	2.52	4.50
9	93.00	92.50
10	93.40	93.40

Source of Variation	M.S.	D.F.	F-ratio	Significant
Method	12.66	1	1.39	
Function	24150.00	9	2650.00	*
Method x Function	13.32	9	1.46	
Within Cell	9.13	180		





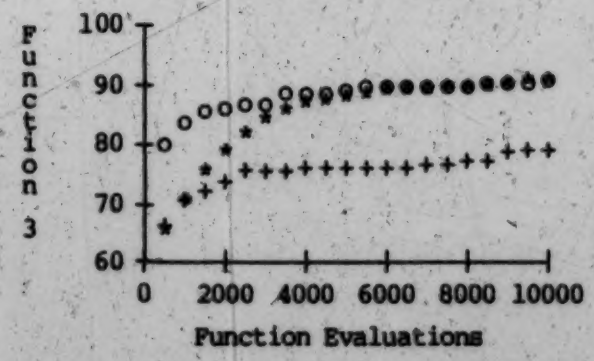
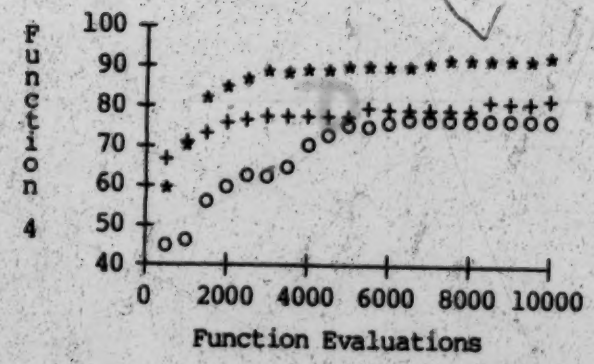
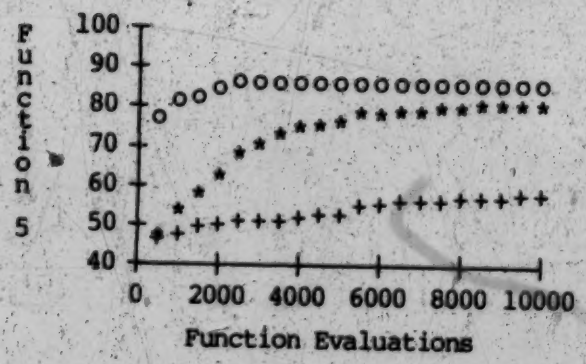


Figure 6.1a. Experiment 9: Global Optimization.
Overall Best Performance on Functions 1, 2, and 3
(+) Pure Random Search
(o) DSC Numerical Method (Convergence Mode)
(*) Genetic Algorithm (Fixed Starts Mode)





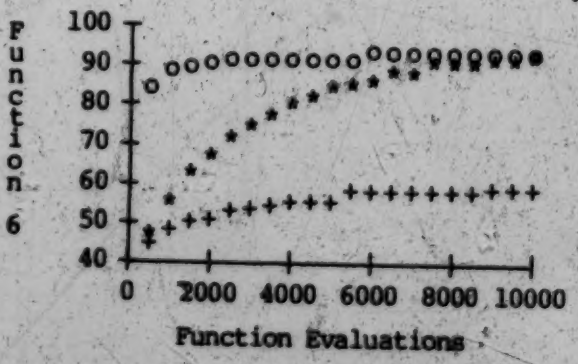


Figure 6.1b. Experiment 9: Global Optimization.
Overall Best Performance on Functions 4, 5, and 6
(+) Pure Random Search
(o) DSC Numerical Method (Convergence Mode)
(*) Genetic Algorithm (Fixed Starts Mode)

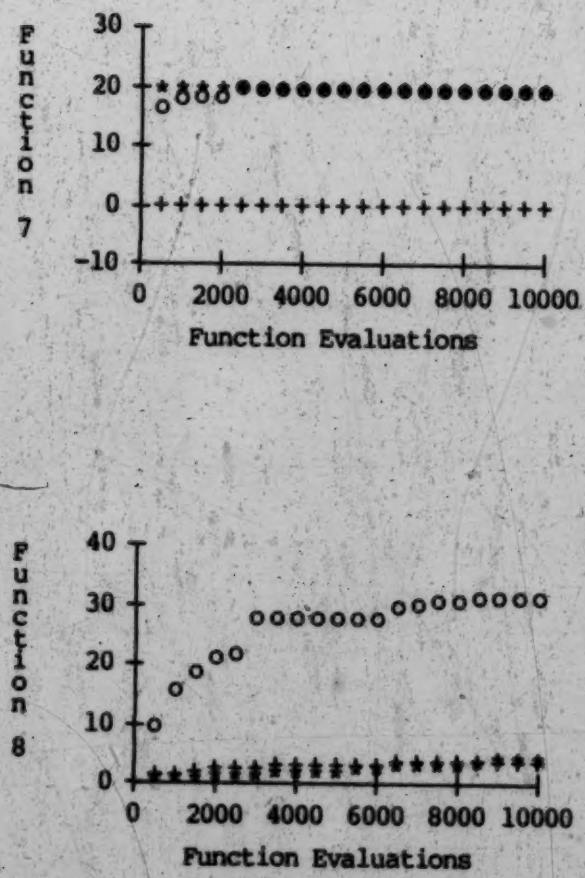
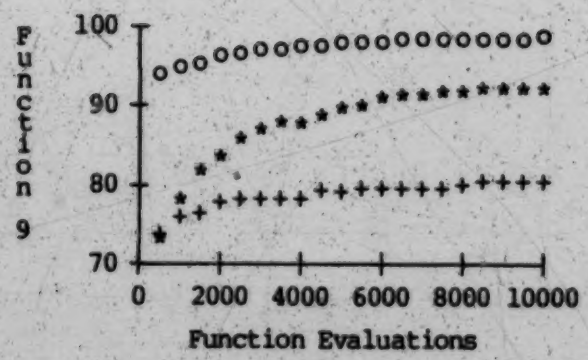


Figure 6.1c. Experiment 9: Global Optimization.
Overall Best Performance on Functions 7 and 8
(+) Pure Random Search
(o) DSC Numerical Method (Convergence Mode)
(*) Genetic Algorithm (Fixed Starts Mode)



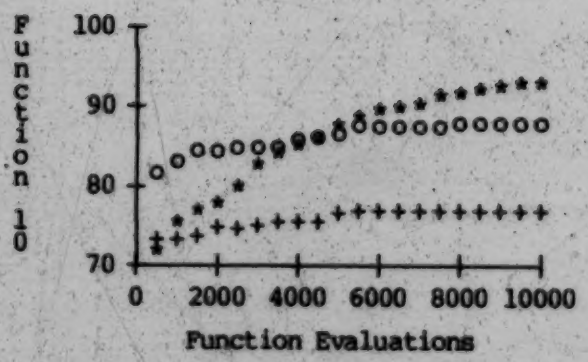


Figure 6.1d. Experiment 9: Global Optimization.
Overall Best Performance on Functions 9 and 10
(+) Pure Random Search
(o) DSC Numerical Method (Convergence Mode)
(*) Genetic Algorithm (Fixed Starts Mode)

The performance curves of DSC differed, depending on the function. As expected, the maximum for Rosenbrock's valley (Function 1) was obtained well before 500 function evaluations. On the other Euclidean unimodal function (Function 9), a good value was also achieved very early. DSC does not handle bounds on the function efficiently. This, and not the discontinuities in the function, may account for its failure to locate the global maximum on Function 9.

On functions with narrow peaks (Functions 2 and 4, possibly Function 8), DSC spent a great deal of time converging on low flat hills, but made sudden, rapid improvements whenever a random start point landed on one of the rarer sharp peaks. Elsewhere, the method quickly climbed those hills that its starting points located, yielding performance curves which paralleled those of PRS, but were higher by values of 10 to 30.

The different rates of improvement of the two methods make comparisons of final values difficult. Tables 6.5 and 6.6, and 6.7 show overall best performance values after 2000, 5000 and 10000 function evaluations, respectively. The functions are grouped according to whether DSC, neither, or GA is superior. Except for Functions 2 and 4 (Shekel's foxholes and the dampened sinusoid), DSC has better early performance (0 to 2000 function evaluations). The genetic algorithm is a good long-term optimizer, failing badly only on Functions 7 and 8.

Table 6.5. Experiment 9: Global Optimization.
Overall Best Performance of DSC versus GA
after 2000 Function Evaluations
(results averaged over 10 runs)

Function	Method	
	DSC(Convergence)	GA(Fixed Starts)
2	95.43	90.05
3	85.88	78.77
5	84.69	62.66
6	91.28	66.99
8	21.37	1.06
9	96.30	83.60
10	84.40	77.80
1	100.00	99.85
7	18.77	20.00
4	60.14	84.89

Source of Variation	M.S.	D.F.	F-ratio	Significant
Method	2022.00	1	60.20	*
Function	18730.00	9	558.00	*
Method x Function	1092.00	9	32.50	*
Within Cell	33.58	180		

Table 6.6. Experiment 9: Global Optimization.
Overall Best Performance of DSC versus GA
after 5000 Function Evaluations
(results averaged over 10 runs)

Function	Method	
	DSC(Convergence)	GA(Fixed Starts)
5	86.10	75.96
6	92.25	84.12
8	27.87	1.90
9	98.00	89.40
1	100.00	99.96
2	98.51	99.79
3	89.09	87.73
7	19.80	20.00
10	86.80	87.60
4	75.13	89.98

Source of Variation	M.S.	D.F.	F-ratio	Significant
Method	688.90	1	37.00	*
Function	19560.00	9	1050.00	*
Method x Function	557.90	9	30.00	*
Within Cell	18.60	180		

Table 6.7. Experiment 9: Global Optimization.
Overall Best Performance of DSC versus GA
after 10000 Function Evaluations
(results averaged over 10 runs)

Function	Method	
	DSC (Convergence)	GA (Fixed Starts)
5	86.10	81.13
8	31.79	4.50
9	98.90	92.50
1	100.00	99.98
2	99.31	99.80
3	90.70	91.13
6	93.22	92.40
7	19.82	20.00
4	77.22	92.41
10	88.40	93.40

Source of Variation	M.S.	D.F.	F-ratio	Significant
Method	165.80	1	10.50	*
Function	19660.00	9	1240.00	*
Method x Function	574.50	9	36.30	*
Within Cell	15.82	180		

6.4 CONCLUSIONS

Both the DSC numerical method and the genetic algorithm are preferable to random search for "black box" global optimization. On functions with very low Euclidean modality, DSC is preferable, since it converges rapidly to optimal values, whether or not the function is smooth. On multimodal functions, both DSC and the genetic algorithm do fairly well.

If the peaks of a multimodal function are narrow (Functions 4 and 10), they are unlikely to be found by DSC. In this case, the genetic algorithm is superior, since the translation it makes to Hamming space for searching may reveal relationships between peaks which are invisible to DSC.

To summarize, the performance of a numerical method is dependent upon the shape of the function in Euclidean space, while a genetic algorithm is affected by the shape of the function in Hamming space. DSC is at its best when optimizing unimodal surfaces in Euclidean space, while a genetic algorithm does best on functions which are linearly independent in Hamming space. If, however, a function has a high degree of nonlinearity in Euclidean space, which is accompanied by nonlinearity in Hamming space, then both methods are inadequate.

CHAPTER 7

CONCLUSIONS

Genetic algorithms are adaptive search methods which employ some of the mechanisms for adaptation in natural evolution. Given the apparent success of natural evolution and the intuitive appeal of adaptive strategies in problem solving, it is natural to anticipate many successful applications for such an algorithm. In Chapter 1, two major questions were posed about these algorithms: what is the domain of problems which genetic algorithms are useful in solving, and which genetic algorithms are desirable for such use. This dissertation has attempted to answer these questions for genetic algorithms applied within the field of function optimization.

In mathematical function optimization, the performance measure which is of primary interest is the overall best solution achieved by a method. For overall best performance, it is desirable to maximize the search capability of the genetic algorithm by maintaining high levels of population variance, usually at the expense of online, or average, performance. Experiment 2 in Chapter 3 indicated that this can be accomplished by using large populations (200 individuals) and choosing a mutation rate inversely proportional to population size.

Population variance also benefits from controlling the error accumulated during parent selection. When parents are sampled for each generation, the sampling method may involve bias or variance. If so, there may be a steady drift over generations away from those populations which would reflect most ideally the concept of selection according to fitness. In Chapter 4 several sampling methods were compared for bias, variance, and effect on genetic algorithm performance. Experiment 4 revealed that population variance is affected by sample variance, which can be minimized by the use of a deterministic, rounding technique for sampling. Good overall best performance can be achieved by the use of either stochastic sampling without replacement or the deterministic method.

It had long been hypothesized that diploidy and dominance could assist in the maintenance of population variance in a genetic algorithm. After experiments 5 through 8 in Chapter 5, however, it must be concluded that diploidy does not raise levels of population variance above those achieved by large haploid populations using minimum variance parent sampling. No major benefits derived from the use of diploidy and any of six dominance schemes in improving algorithm performance.

This result is better understood in light of developing theories in population genetics on the causes and purposes of dominance. Diploidy may be beneficial in natural organisms as a defense against deleterious mutations, and

may improve organisms in situations of heterozygote superiority or missing optimal alleles. None of these benefits would apply to genetic algorithms for function optimization which use binary chromosomes.

The question of where genetic algorithms are applicable in static function optimization can now be answered. In Chapter 6 a set of ten test functions were designed which incorporated a number of levels of dimensionality, modality, and continuity. On these, a genetic algorithm was compared to random search and a "direct search" numerical method. Although genetic algorithms are superior to random search for global function optimization, they cannot be considered any more robust than a good local numerical method run in global mode. Such numerical methods are designed for functions which are unimodal in Euclidean space. They can be designed to handle discontinuity, high dimensionality, and the absence of explicit gradient information. Multimodality, however, invariably impairs their performance.

Likewise, the performance of genetic algorithms is not affected by continuity, dimensionality, or differentiability. In addition, genetic algorithms are capable of optimizing some highly multimodal functions. What does affect these algorithms are the nonlinearities of a function when transformed into an equivalent function in Hamming space. Genetic algorithms perform well on functions which are linearly independent in Hamming space;

nonlinearities among many loci impair their performance.

Thus genetic algorithms with binary chromosomes are useful on any function where it is suspected that performance is determined by the individual loci of a binary number or set. That is, in order for a genetic algorithm to be effective, there must be not only a convenient binary representation for solutions, but the elements of this representation must have a straightforward effect on the function value. This is not usually the case when binary chromosomes are translated into real x-values for evaluation in a difficult function in Euclidean space. Functions which are multimodal but smooth in Euclidean space, that is, two neighboring points in Euclidean space have close values, are probably best optimized by descent methods.

This research has concentrated on genetic algorithms with binary chromosomes applied to unconstrained function optimization. Many other applications for genetic algorithms remain to be studied, some of which will probably require alternative representations of solutions.

First, there are constrained problems in function optimization. One method for handling constraints is to give a solution which does not satisfy the constraints a very poor value and include it in the population. Also, it may be possible to design a measure of how far outside the constraints a solution falls. Then a separate population of invalid solutions could be maintained in order to use the "better" invalid solutions for breeding.

Even more interesting are set problems. As mentioned in Section 2.3.5, if the solution space of a problem consists of subsets of one superset, the obvious representation is binary loci which indicate inclusion of set elements in the solution subset. It has been shown that genetic algorithms are best used when the evaluation of a solution depends upon individual values in a natural binary representation. Set problems present a promising area of application for genetic algorithms. However, the simplest representation is not feasible if the superset is very large. Even if the superset is small, this representation may not lead to good genetic algorithms.

Consider the knapsack problem, one of a class of computationally complex set problems (Aho, Hopcroft, and Ullman, 1974). The problem is the following: given a positive integer M and a set Q of pairs of positive integers, $Q = \{(p_1,c_1),\ldots,(p_n,c_n)\}$,

maximize $$P = \sum_{i=1}^{n} \delta_i p_i$$

such that $$\sum_{i=1}^{n} \delta_i c_i \leq M$$

and $$\delta_i \in \{0, 1\}, \text{ for } i=1,\ldots,n.$$

Here the representation mentioned above for problems of size n = 100 will require chromosomes of length 100. It might be expected that a genetic algorithm using such a representation would form a valuable heuristic, supplying approximate solutions in linear time. However, many

specific heuristics have been developed for this problem and others like it (Korte, 1979). Comparisons reveal that the heuristic method of Ibarra and Kim (1974; 1975) can achieve far better solutions than the genetic algorithm in less than one tenth the time.

It seems likely that for most difficult set problems, the successful application of genetic algorithms will depend upon innovative representations and corresponding changes to the operators within the algorithms. In addition, research on such problems would probably benefit from the comparison of genetic algorithm performance to the performance of a numerical method such as that of Davies, Swann, and Campey (Swann, 1964) adapted for use in Hamming space.

Genetic algorithms for function optimization emphasize overall best performance in the optimization of environments characterized by a priori ignorance and static complexity. Since the mechanisms of genetic algorithms are taken from natural systems, which operate within environments of dynamic ignorance and uncertainty, genetic algorithms may prove to be most valuable on time-varying or probabilistic environments. If so, measures such as online performance will be of primary importance. In particular, genetic algorithms for tracking time-varying environments have not yet been thoroughly analyzed. Haploid and diploid genetic algorithms have been compared on time-varying environments. The haploid algorithm exhibited good online performance, but has not yet been tested against other, non-genetic methods.

REFERENCES

Aho, A.V., Hopcroft, J.E., Ullman, J.D. 1974. The Design and Analysis of Computer Algorithms. Addison-Wesley Publishing Company, Reading, Massachusetts.

Anderssen, R.S. 1972. Global optimization. In Optimization. Anderssen, R.S., Jennings, L.S., Ryan, D.M., editors. University of Queensland Press, St. Lucia, Queensland. 26-48.

Bagley, J.D. 1967. The Behavior of Adaptive Systems Which Employ Genetic and Correlation Algorithms. Ph.D. Dissertation, University of Michigan, Ann Arbor.

Box, M.J., Davies, D., Swann, W.H. 1969. Non-Linear Optimization Techniques. ICI Monograph Number 5, Oliver and Boyd, Edinburgh.

Brenner, M.E. 1963. Selective sampling--a technique for reducing sample size in simulation of decision making problems. Journal of Industrial Engineering 14: 291-296.

Brent, R.P. 1972. A new algorithm for minimizing a function of several variables without calculating derivatives. In Optimization. Anderssen, R.S., Jennings, L.S., Ryan, D.M., editors. University of Queensland Press, St. Lucia, Queensland. 14-25.

Brooks, S.H. 1958. A discussion of random methods for seeking maxima. Operations Research 6: 244-251.

Burt, J.M., Gaver, D.P., Perlas, M. 1970. Simple stochastic networks: some problems and procedures. Naval Research Logistics Quarterly 17: 439-459.

Cavicchio, D.J. 1970. Adaptive Search Using Simulated Evolution. Ph.D. Dissertation, University of Michigan, Ann Arbor.

Clark, B. 1964. Frequency-dependent selection for the dominance of rare polymorphic genes. Evolution 18: 364-369.

Cooper, L., Steinberg, D. 1970. Introduction to Methods of Optimization. W.B. Saunders Company, Philadelphia.

Crosby, J.L. 1963. The evolution and nature of dominance. Journal of Theoretical Biology 5: 35-51.

Crow, J.F., Kimura, M. 1970. Introduction to Population Genetics Theory. Harper and Row, New York.

De Jong, K.A. 1975. Analysis of the Behavior of a Class of Genetic Adaptive Systems. Ph.D. Dissertation, University of Michigan, Ann Arbor.

Downham, D.Y., Roberts, F.D.K. 1967. Multiplicative congruential pseudo-random number generators. The Computer Journal 10: 74-77.

Ehrenfeld, S., Ben-Tuvia, S. 1962. The efficiency of statistical simulation procedures. Technometrics 4: 257-275.

Ewens, W.J. 1968. Population Genetics. Methuen and Company, London.

Felsenstein, J. 1974. The evolutionary advantage of recombination. Genetics 78: 737-756.

Fisher, R.A. 1958. The Genetical Theory of Natural Selection. Dover Publications, New York.

Fletcher, R. 1965. Function minimization without evaluating derivatives--a review. The Computer Journal 8: 33-41.

Fletcher, R. 1969. A review of methods for unconstrained optimization. In Optimization. Fletcher, R., editor. Academic Press, New York. 1-12.

Fletcher, R., Powell, M.J.D. 1963. A rapidly convergent descent method for minimization. The Computer Journal 6: 163-168.

Ford, E.B., Sheppard, P.M. 1965. Natural selection and the evolution of dominance. Heredity 21: 139-146.

Frantz, D.R. 1972. Non-Linearities in Genetic Adaptive Search. Ph.D. Dissertation, University of Michigan, Ann Arbor.

Hammersley, J.M., Handscomb, D.C. 1964. Monte Carlo Methods. John Wiley and Sons, New York.

Hill, J.D. 1969. A search technique for multimodal surfaces. IEEE Transactions on Systems Science and Cybernetics SSc-3: 2-7.

Holland, J.H. 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.

Hollstien, R.B. 1971. Artificial Genetic Adaptation in Computer Control Systems. Ph.D. Dissertation, University of Michigan, Ann Arbor.

Hood, L.E., Wilson, J.H., Wood, W.B. 1975. Molecular Biology of Eucaryotic Cells. W.A. Benjamin, Menlo Park, Calif.

Ibarra, O.H., Kim, C.E. 1974. Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems. Technical Report 74-13, University of Minnesota, Minneapolis.

Ibarra, O.H., Kim, C.E. 1975. Fast approximation algorithms for the knapsack and sum of subset problems. Journal of the Association for Computing Machinery 22: 463-468.

International Mathematical and Statistical Libraries, Inc. 1975. IMSL Library 1. Houston.

Jacoby, S.L.S., Kowalik, J.S., Pizzo, J.T. 1972. Iterative Methods for Non-Linear Optimization Problems. Prentice-Hall, Englewood Cliffs, New Jersey.

Jarvis, R.A. 1975. Optimization strategies in adaptive control: a selective survey. IEEE Transactions on Systems, Man and Cybernetics SMC-5: 83-94.

Kahn, H., Marshall, A.W. 1953. Methods of reducing sample size in Monte Carlo computations. Operations Research 1: 263-278.

Kimura, M. 1968. Genetic variability maintained in a finite population due to a mutation production of neutral and nearly neutral isoalleles. Genetical Research 11: 247-269.

Kleijnen, J.P.C. 1974. Statistical Techniques in Simulation, Part I. Marcel Dekker, New York.

Korte, B. 1979. Approximative algorithms for discrete optimization problems. In Discrete Optimization I. Hammer, P.L., Johnson, E.L., Korte, B.H., editors. North Holland Publishing Company, New York. 85-120.

Kowalik, J., Osborne, M.R. 1968. Methods for Unconstrained Optimization Problems. Elsevier, New York.

Li, W.H. 1978. Maintenance of genetic variability under the joint effect of mutation, selection and random drift. Genetics 90: 349-382.

Martin, N. 1973. Convergence Properties of a Class of Probabilistic Adaptive Schemes Called Sequential Reproductive Plans. Ph.D. Dissertation, University of Michigan, Ann Arbor.

Maynard Smith, J. 1978. The Evolution of Sex. Cambridge University Press, Cambridge.

Nelder, J., Mead, R. 1965. A simplex method for function minimization. The Computer Journal 7: 308-313.

Powell, M.J.D. 1964. An efficient method for finding the minimum of a function of several variables without calculating derivatives. The Computer Journal 7: 155-162.

Price, W.L. 1977. A controlled random search procedure for global optimization. The Computer Journal 20: 367-370.

Rosenbrock, H.H. 1960. An automatic method for finding the greatest or least value of a function. The Computer Journal 3: 175-184.

Sampson, J.R. 1976. Adaptive Information Processing. Springer-Verlag, New York.

Samuel, A.L. 1959. Some studies in machine learning using the game of checkers. IBM Journal of Research and Development 3: 211-299.

Shekel, J. 1971. Test functions for multimodal search techniques. Proceedings of the Fifth Annual Princeton Conference on Information Science and Systems. Princeton, New Jersey.

Stewart, G.W. 1967. A modification of Davidon's minimization method to accept difference approximations of derivatives. Journal of the Association for Computing Machinery 14: 72-83.

Strickberger, M.W. 1976. Genetics. The MacMillan Company, New York.

Swann, W.H. 1964. Report on the Development of a New Direct Search Method of Optimization. ICI Limited, Central Instrument Laboratory Research Note 64/3.

Watterson, G.A. 1977. Heterosis or neutrality? Genetics 85: 789-814.

Wetzel, A. 1979. Personal communication.

Winer, B.J. 1962. Statistical Principles in Experimental Design. McGraw-Hill Book Company, New York.

APPENDIX A

FUNCTION OPTIMIZATION GLOSSARY

Bounds: The bounds of a function are maximum or minimum values for each x-value of the function. These provide outer limits on acceptable solutions. Constraining equations of the forms $x_i < m$, $x_i > m$, $x_i \leq m$ and $x_i \geq m$ define bounds.

Concavity: A function is strictly upward concave if the line segment joining any two points on the function falls on or below the function.

Constraints: A function is constrained if solutions must satisfy conditions other than simple bounds. Additional equations, each involving several dimensions, are a common example of constraints.

Continuity: A function f is continuous at a point $P = (p_1,\ldots,p_n)$ if it is defined at that point and, for $X = (x_1,\ldots,x_n)$, $\lim_{X \to P} f(X) = f(P)$.

Convexity: A function is strictly upwards convex if the line segment joining any two points on the function lies on or above the function.

Dependence: see Linear Independence.

Descent Methods: see Numerical Function Optimization Methods.

Differentiability: A function f is differentiable at a point $X = (x_1,x_2,\ldots,x_n)$ if, for $i=1,\ldots,n$,

$\lim_{h\to 0}(f(x_1,\ldots,x_i+h,\ldots,x_n)-f(x_1,\ldots,x_i,\ldots,x_n))/h$ exists.

Dimensionality: Dimensionality refers to the number of dimensions in the Euclidean space over which a function is defined.

Direct Search Methods: see Numerical Function Optimization Methods.

Euclidean space: The notation $(x_1,\ldots,x_n)$ used for a solution point is a representation of that point in n-dimensional Euclidean space. Two points are close together in Euclidean space if the differences between their respective x-values on each dimension are small.

Gradient Methods: see Numerical Function Optimization Methods.

Hamming space: A discrete binary representation of a solution point is a representation of that point in Hamming space. Two points are adjacent in Hamming space if they differ by only one bit (0 or 1). Space here refers not to a mathematical vector space, but simply to an encoded set of points with a distance measure.

Independence: see Linear Independence.

Indirect Methods: Indirect optimization methods are methods which do not employ search. The optimum is computed without evaluating individual points of the function. Solving a linear equation and its linear constraint equations by elimination is an indirect method. Computing the critical points of a function by finding points where the function derivatives are zero is also indirect.

Linear Independence: A function is linearly independent over a domain of variables if it can be written as the sum of subfunctions, each of which has a domain of at most one variable. $f(x,y) = 3x + y^2$ is linearly independent over x and y. A function which cannot be reduced to such a sum has nonlinearities, or dependencies, between two or more variables. $f(x,y,z) = x + 2x^2y + z^3$ has two dependent variables, x and y.

Linearity: see Linear Independence.

Modality: Modality refers to the number of peaks (in maximization problems) or valleys (in minimization problems) of a function. Let x_{opt} be the location of the global maximum of a function f. Assume f is to be maximized. Let x and y be other points for which f is defined.

Unimodal functions have one relative maximum value: there exists a path x to x_{opt} for which f is nondecreasing.

Strictly unimodal functions have no plateaus, but may have ridges: there exists a path x to x_{opt} for which f is increasing.

Strongly unimodal functions have only straight ridges: f is increasing on the line segment between x and x_{opt}.

Linearly unimodal functions have no ridges, but may have plateaus: f is unimodal on the line segment between any x and y.

Nonlinearity: see Linear Independence.

Numerical Function Optimization Methods:

Gradient methods (also called Descent methods) use information on derivatives to locate the direction of greatest change of the function. New points are located by moving in the direction of greatest improvement from previous points.

Direct Search methods do not employ gradient information. There are three main types: Tabulation methods, which "box in" the optimal point and then reduce the size of the box, Sequential methods, which repeatedly evaluate points at the vertices of geometrical figures, and Linear methods, which use a set of direction vectors and repeatedly perform linear searches along those directions.

Tabulation Methods: see Numerical Function Optimization Methods.

APPENDIX B

BIAS OF REMAINDER STOCHASTIC SAMPLING WITHOUT REPLACEMENT

Proposition: Sampling without replacement from the r_i, $i=1,\ldots,n$, is almost always biased.

Proof:

The stochastic portion of remainder stochastic sampling is altered as follows to yield sampling without replacement (see Figure 4.3). If i is sampled, r_i ("fraction(i)") is replaced by 0 in the distribution and the range R for the random numbers is reduced by r_i.

$$\begin{aligned} E[f_i] &= E[w_i + g_i] \\ &= w_i + E[g_i]. \\ E[g_i] &= 0(P(g_i=0) + 1(P(g_i=1)) \\ &= P(g_i=1). \end{aligned}$$

Let z_j represent the r value of the element sampled at time j. Then the probability of selecting element i is equal to the sum of the probabilities of selecting i at each time j, $j=1,\ldots,R$, or

$$\begin{aligned} P(g_i=1) &= \frac{r_i}{R} + (1 - \frac{r_i}{R})\frac{r_i}{R-z_1} + \ldots + \\ &\quad (1 - \frac{r_i}{R})(1 - \frac{r_i}{R-z_1})\ldots(1 - \frac{r_i}{R-z_1-\ldots-z_{R-2}})\frac{r_i}{R-z_1-\ldots-z_{R-1}} \\ &= \frac{r_i}{R}[1 + \frac{(R-r_i)}{(R-z_1)} + \frac{(R-r_i)(R-r_i-z_1)}{(R-z_1)(R-z_1-z_2)} \\ &\quad + \ldots + \frac{(R-r_i)(R-r_i-z_1)}{(R-z_1)(R-z_1-z_2)}\cdots\frac{(R-r_i-z_1-\ldots-z_{R-2})}{(R-z_1-z_2-\ldots-z_{R-1})}]. \end{aligned}$$

Each of the terms inside the brackets is less than or equal to one. If the non-zero r_i's are not all equal, then for

the first element (the one which has the maximum r value), at least one of the terms must be strictly less than one. Hence for i = 1,

$$P(g_1=1) < \frac{r_1}{R}[R]$$
$$< r_1.$$

Thus

$$E[f_1] < w_1 + r_1$$
$$< e_1.$$

In general, as long as the non-zero r_i are not all equal (a finite subset of the possible cases), the elements with larger r values will be biased towards underselection and those with smaller values towards overselection.

APPENDIX C

MEAN SQUARE ERROR OF THREE SAMPLING METHODS

Assume the notation of Chapter 4. Thus for i=1,...,n,

$$e_i = mp_i$$

$$w_i = \lfloor e_i \rfloor$$

$$r_i = e_i - w_i$$

$$W = \sum_{i=1}^{n} w_i$$

$$R = \sum_{i=1}^{n} r_i .$$

Then $W + R = m$.

Let the r_i be ordered so that $r_i \geq r_{i+1}$, i=1,...,n-1.

$$MSE_S = \sum_{i=1}^{n} mp_i(1 - p_i) = \sum_{i=1}^{n} e_i(1 - \frac{e_i}{m})$$

$$MSE_{RS} = \sum_{i=1}^{n} r_i(1 - \frac{r_i}{R}) = R - \frac{1}{R}\sum_{i=1}^{n} r_i^2$$

$$MSE_D = \sum_{i=1}^{R} (1 - r_i)^2 + \sum_{i=R+1}^{n} r_i^2 = R - 2\sum_{i=1}^{R} r_i + \sum_{i=1}^{n} r_i^2$$

The differences between the mean square errors for the three methods are examined by breaking each comparison into cases. This sheds light on which conditions render the values equal and which provide strict inequality.

Proposition: $MSE_S \geq MSE_{RS}$.

Proof:

Case I: $R = 0$.

$MSE_{RS} = 0$

$\therefore MSE_S > MSE_{RS}$.

Case II: $1 \leq R < m$.

Consider each $A_i = e_i(1 - \frac{e_i}{m}) - r_i(1 - \frac{r_i}{R})$, $i=1,\ldots,n$.

A. If $e_i = 0$, $A_i = 0$.

B. If $e_i > 0$,

1. If $w_i = 0$, $e_i = r_i$.

$$A_i = r_i(1 - \frac{r_i}{m}) - r_i(1 - \frac{r_i}{R})$$
$$= r_i^2(\frac{1}{R} - \frac{1}{m})$$
$$> 0.$$

2. If $w_i \geq 1$,

a. If $r_i = 0$, $e_i = w_i$.

$$A_i = w_i(1 - \frac{w_i}{m}).$$

Since $s \geq 1$, $w_i < m$, so

$A_i > 0$.

b. If $r_i > 0$,

$$A_i = \frac{1}{Rm}(Rme_i - Re_i^2 - Rmr_i + mr_i^2).$$

Since $m \geq w_i + R$,

$$A_i \geq \frac{1}{Rm}[(w_i + R)\cdot Rw_i - R(w_i^2 + 2w_ir_i + r_i^2) + (w_i + R)\cdot r_i^2]$$
$$= \frac{1}{Rm}(R^2w_i - 2Rw_ir_i + w_ir_i^2)$$
$$= \frac{1}{Rm}(w_i)(R - r_i)^2.$$

$s \geq 1$ and $r_i < 1$, so

$A_i > 0$.

$\therefore MSE_S > MSE_{RS}$.

Case III: R=m.

This occurs only when $e_i = r_i$ for i=1,...,n, in which case the two methods are identical. Therefore

$MSE_S = MSE_{RS}$.

Summary: $MSE_S \geq MSE_{RS}$.

In particular, $MSE_S = MSE_{RS}$ only if $e_i < 1$ for i=1,...,n. This is possible only if m < n. Otherwise

$MSE_S > MSE_{RS}$.

Proposition: $MSE_{RS} \geq MSE_{D}$.

Proof:

Case I: R = 0.

In this case, the deterministic and remainder stochastic methods are identical.

$\therefore MSE_{RS} = MSE_{D}$.

Case II: R = 1.

A. Suppose $r_i = \frac{1}{x}$ for some $1 < x \leq n$ and $i=1,\ldots,x$, and $r_i = 0$ for $i=x+1,\ldots,n$.

Then

$$MSE_{RS} = \sum_{i=1}^{x} \frac{1}{x}(1 - \frac{1}{x})$$

$$= 1 - \frac{1}{x}.$$

$$MSE_{D} = (1 - \frac{1}{x})^2 + \sum_{i=2}^{x} (\frac{1}{x})^2 + \sum_{i=x+1}^{n} 0$$

$$= 1 - \frac{2}{x} + (\frac{1}{x})^2 + (x - 1)(\frac{1}{x})^2$$

$$= 1 - \frac{1}{x}.$$

$\therefore MSE_{RS} = MSE_{D}$.

B. Suppose $r_i \neq r_j$ for some $r_i \neq 0$, $r_j \neq 0$.

$$MSE_{RS} = 1 - \sum_{i=1}^{n} r_i^2$$

$$MSE_{D} = 1 - 2r_1 + \sum_{i=1}^{n} r_i^2$$

$$\therefore MSE_{RS} - MSE_{D} = 2r_1 - 2\sum_{i=1}^{n} r_i^2$$

$$= 2[r_1(1-r_1) - \sum_{i=2}^{n} r_i^2].$$

Since $r_1 + \sum_{i=2}^{n} r_i = 1$,

$$MSE_{RS} - MSE_D = 2(r_1 \sum_{i=2}^{n} r_i - \sum_{i=2}^{n} r_i^2)$$

$$= 2 \sum_{i=2}^{n} r_i (r_1 - r_i).$$

Now $r_1 \geq r_i$ for $i=2,\ldots,n$. By the condition that there exist $r_i \neq r_j$ where neither is zero, $r_1 > r_j$ for some $r_j \neq 0$.

$\therefore MSE_{RS} - MSE_D > 0$ or

$MSE_{RS} > MSE_D$.

Case III: $2 \leq R \leq m$.

$$MSE_{RS} - MSE_D = R - \frac{1}{R} \sum_{i=1}^{n} r_i^2 - R + 2 \sum_{i=1}^{R} r_i - \sum_{i=1}^{n} r_i^2$$

$$= 2 \sum_{i=1}^{R} r_i - (1 + \frac{1}{R}) \sum_{i=1}^{n} r_i^2$$

$$> 2[\sum_{i=1}^{R} r_i - \sum_{i=1}^{n} r_i^2]$$

$$= 2[\sum_{i=1}^{R} r_i (1 - r_i) - \sum_{i=R+1}^{n} r_i^2]$$

$$\geq 2[\sum_{i=1}^{R} r_R (1 - r_i) - \sum_{i=R+1}^{n} r_i^2]$$

$$= 2[r_R (R - \sum_{i=1}^{R} r_i) - \sum_{i=R+1}^{n} r_i^2]$$

$$= 2[r_R \sum_{i=R+1}^{n} r_i - \sum_{i=R+1}^{n} r_i^2]$$

$$= 2 \sum_{i=R+1}^{n} r_i (r_R - r_i)$$

$$\geq 0$$

$\therefore MSE_{RS} > MSE_{D}$.

Summary: $MSE_{RS} \geq MSE_{D}$.

In particular, $MSE_{RS} = MSE_{D}$ only if

$R = 0$ or

$R = 1$ and $r_i = r_j$ for all r_i and $r_j \neq 0$.

Otherwise $MSE_{RS} > MSE_{D}$.

APPENDIX D

GLOBAL OPTIMIZATION METHODS

(This appendix is not intended to supply full descriptions of the algorithms employed. Only those parameters which are specific to the implementation are mentioned.)

PRS (pure random search). 10000 uniform random points in the space are evaluated.

CRS_c (creeping random search, convergence mode). CRS evaluates 30 points which are normally distributed around a randomly chosen initial point, with standard deviation $0.2 \cdot (U - L)$, where U and L are the upper and lower bounds of x_1. The center of the distribution is then moved to the best point found and the square root of the distance moved is adopted as the new standard deviation. The distribution continues to move every 30 evaluations until two moves of distance less than 2.e-10 (0.02 for Function 9, 2 for Function 10) occur. Then the process repeats from a new random point.

CRS_f (creeping random search, fixed starts mode). If CRS were run on 200 starting points for a total of 10000 function evaluations, there would be only one movement of the distribution from each point. So CRS_f employs 100 starting points, allowing 100 function evaluations from each point. The other features of the method are described under CRS_c.

$\underline{DSC}_c$ (Davies, Swann, and Campey method, convergence mode). DSC performs a linear optimization in each of the d directions in the space, using a random starting point and an initial step size of $0.1 \cdot (U_i - L_i)$ in each direction. U_i and L_i are the upper and lower bounds of each dimension i. Then a new set of d orthogonal directions is chosen to reflect the direction of greatest improvement in the previous iteration, and a new iteration of linear optimizations is performed. When one iteration yields a best point less than 2.e-10 (0.02 for Function 9, 2 for Function 10) from its initial point, another random starting point is selected and the process repeats.

$\underline{DSC}_f$ (Davies, Swann, and Campey method, fixed starts mode). DSC searches for 50 function evaluations on each of 200 random starting points. The iterations of the method are explained under DSC_c.

$\underline{GA}_c$ (genetic algorithm, convergence mode). A haploid genetic algorithm with population size 200, crossover rate 0.6, mutation rate 0.001, and deterministic parent selection is run repeatedly to convergence on random initial populations. Convergence for CRS and DSC is defined as the limiting of search to a region of radius 2.e-10 (0.02 for Function 9, 2 for Function 10) around a point. Since the genetic algorithm is searching with a resolution of 1.e-10 (0.01 for Function 9, 1 for Function 10) between points, a definition of convergence similar to that of the other methods would be the reduction of search to within a region

of radius 2 about a point in Hamming space. Therefore, the genetic algorithm will be assumed to have converged in its search if all but two loci have reached 90% convergence in the population.

$\underline{GA}_f$ (genetic algorithm, fixed starts mode). The haploid genetic algorithm described under GA_c is run on one randomly selected starting population of size 200 for 10000 function evaluations.